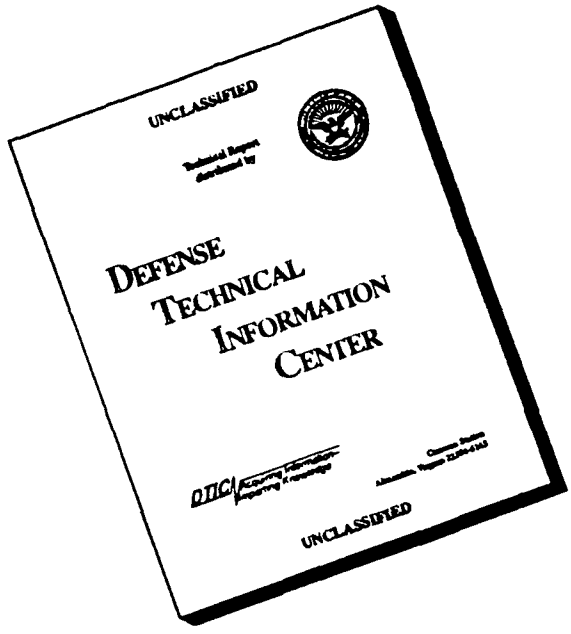
Technical Report 1564
October 1992

High-Performance Visualization Applied to Computational Electromagnetics

Method-of-Moments on a Silicon Graphics Workstation

L. C. Russell
J. W. Rockway

DISCLAIMER NOTICE



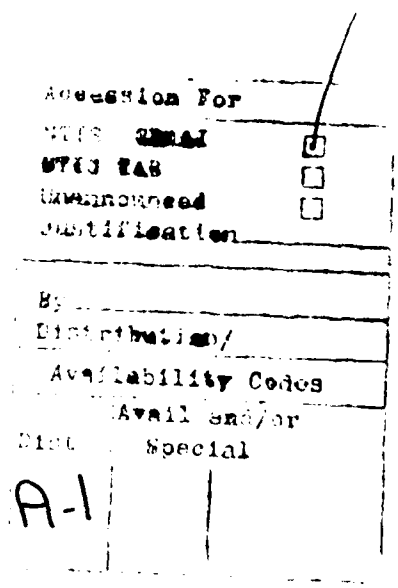
THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Technical Report 1564
October 1992

High-Performance Visualization Applied to Computational Electromagnetics

Method-of-Moments on a Silicon Graphics Workstation

L. C. Russell
J. W. Rockway



NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

This work was performed by the Electromagnetic Technology and Systems Branch under the Independent Exploratory Development Program, OCNR-10P, Arlington, VA 22217.

Released by
J. B. Rhode, Head
Electromagnetic Technology
and Systems Branch

Under authority of
R. J. Kochanski, Head
Communications Systems
Engineering and Integration
Division

SM

EXECUTIVE SUMMARY

OBJECTIVES

Demonstrate the value of high-performance visualization techniques for the method-of-moment modeling of electromagnetic field radiation and scattering for antennas in a complex environment. Improve the utility of computational techniques used in electromagnetic analysis, with special emphasis on ship antenna design.

APPROACH

Method-of-moments is one of the most commonly used techniques in computational electromagnetics. It is especially useful for many of the frequencies of interest in the electromagnetic topside design of Navy ships. The method-of-moments technique involves solving a set of coupled integral equations to determine the currents on a structure.

Computational electromagnetics can be thought of as a three-step process: problem definition, computation, and solution description. The drawback to using a computational technique such as method-of-moments is in the effort required at each of the three steps. Input models must be extensively validated before calculations are performed. Enormous quantities of output data are generated, including currents on all wire segments, near-field contours, and far fields. Advanced visualization techniques can assist in input validation and the rapid interpretation of output data.

The computational electromagnetics code used was the Numerical Electromagnetics Code - Method of Moments (NEC-MoM). Electromagnetic visualization was performed on a Silicon Graphics Incorporated (SGI) 4D/320GTXB workstation.

CONCLUSIONS

Visualization tools now exist for performing analysis and validation at all three steps of a method-of-moments design procedure. These tools have greatly improved the utility of computational techniques used in electromagnetic analysis and are now being used to support topside ship design projects for the Navy.

CONTENTS

EXECUTIVE SUMMARY iii

1.0 INTRODUCTION 1

2.0 COMPUTATIONAL ELECTROMAGNETICS 1

2.1 ELECTROMAGNETIC MODEL 1

2.2 NEC-MoM 2

2.3 NEED FOR VISUALIZATION 3

3.0 VISUALIZATION TECHNIQUES 3

3.1 COLOR CODING 3

3.2 COMPLEX DATA 4

3.3 VECTOR DATA 4

4.0 APPROACH 4

4.1 SGI 4D/320 5

4.2 SOFTWARE PACKAGES 5

4.3 NEC DATA SETS 6

5.0 VISUALIZATION PROGRAM STRUCTURE 6

5.1 WINDOWS 6

5.2 MENUS 6

5.3 TRANSFORMATIONS 7

5.3.1 Zoom 7

5.3.2 Rotate 7

5.3.3 Translate 8

5.4 PICK CORRELATION 8

5.5 USER'S GUIDE 9

6.0 GEOMETRY DESCRIPTION PRODUCTS 10

7.0 SOLUTION DESCRIPTION PRODUCTS 10

7.1 CURRENTS 11

7.2 NEAR FIELDS 11

7.3 FAR FIELDS 12

8.0 MoM MATRIX DISPLAY 12

8.1 EXPLORER MAP 12

8.2 GRAPHICS LIBRARY PROGRAM 12

9.0 OBSERVATIONS 13

9.1 VIDEO 14

9.2 PRODUCTION CODE 14

9.3 PC IMPLEMENTATION 14

9.4 OTHER CEM CODES 14

10.0 CONCLUSIONS 14

11.0 REFERENCES 15

APPENDICES:

A. Source Code for *filter* A-1

B. Source Code for *view_nec* B-1

C. Source Code for *view_mom* C-1

FIGURES

1. Electromagnetic model. 2

2. Wire parameters window. 8

3. *view_nec* wire segmentation display window. 10

4. MoM matrix 2D display for a four-sided loop. 13

TABLE

1. HERP and HERO near-field values 11

1.0 INTRODUCTION

This report presents the findings of a Navy-funded Independent Exploratory Development (IED) investigation. This investigation applied advanced visualization techniques to an existing computational electromagnetic (CEM) code. The goal was to demonstrate that high-performance visualization can improve the utility of computational techniques used in ship electromagnetic designs.

The CEM code used was the Numerical Electromagnetics Code - Method of Moments (NEC-MoM). The NEC-MoM code was not run on the visualization workstation. Instead, it was run on a high-performance computing (HPC) platform, a Convex C-220, with only the NEC-MoM input and selected output files ported to a Silicon Graphics, Incorporated (SGI) machine for pre- and postprocessing visualization.

The visualization platform used was an SGI 4D/320GTXB, representing the most sophisticated visualization hardware available. Two visualization codes were developed as a result of this project: *view_nec* and *view_mom*. The programming was done using the C language calling SGI Graphics Library™ (GL) functions.

This report is organized along the following lines: Chapter 2 describes computational electromagnetics, the method-of-moments technique, and the need for visualization. Chapter 3 discusses techniques for visualizing data. Chapter 4 discusses the approach used for this IED project, including hardware and software options. Chapters 5, 6, 7, and 8 give detailed descriptions of the visualization products that were developed as a result of this project. Chapters 9 and 10 provide observations and conclusions.

2.0 COMPUTATIONAL ELECTROMAGNETICS

Computational electromagnetics is that branch of electromagnetics that routinely involves using a computer to obtain results. It is a complementary tool to the classical techniques of experimental observation and mathematical analysis.

The CEM application of interest to the authors of this report is naval ship electromagnetic design. The specific area of interest is antenna design. Key technical parameters for antenna performance include impedance, near fields, coupling, and far-field radiation patterns.

2.1 ELECTROMAGNETIC MODEL

The electromagnetic model uses transfer functions derived from Maxwell's equations (figure 1). The inputs to the transfer functions include an environmental description of the problem as well as the specified input. The environmental description is defined by both the electrical and the geometrical properties of the structures and the space in which they reside. The input is the specified excitation, which can be either a voltage source applied to an antenna or a plane wave impeding on the defined structure. The outputs from the transfer functions are the induced currents on the structures or the impedance of the antenna. These induced currents can be used to determine both the near and far fields.

For naval ship electromagnetic design the near fields are of interest in determining coupling between antennas as well as hazards to personnel, fuel, and ordnance. Far fields are used to determine the performance of the electromagnetic system.

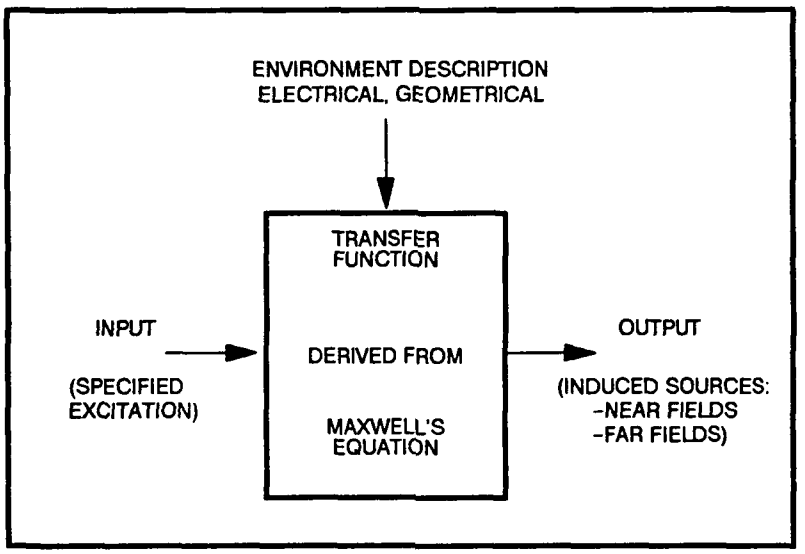


Figure 1. Electromagnetic model.

2.2 NEC-MoM

The numerical electromagnetic code was developed for antenna modeling. NEC includes NEC-MoM, NEC-basic scattering, and NEC-reflector antenna codes (Li, Logan, and Rockway, 1988). All these codes have been validated and extensively documented. NEC-MoM is a development effort of the Lawrence Livermore National Laboratory (Burke and Pogio, 1981; Burke, 1989).

To model complex structures, NEC-MoM uses an electric-field integral equation (EFIE) for wires and a magnetic-field integral equation (MFIE) for surfaces. The EFIE is well suited for thin wires, while the MFIE is attractive for modeling smooth surfaces that bound large closed volumes. The EFIE can also be used to model surfaces, by using a wire grid to represent a surface. The EFIE wire grid is preferred for surfaces that are not smooth and that enclose volumes that are not large. For the task described in this report, only the EFIE technique was used.

The thin-wire approximation is applied to the EFIE to reduce the equation to a scalar integral equation. Transverse currents and transverse variation of the axial current on the wire are neglected. The boundary condition on an electric field is enforced in the axial direction only. The approximations made in NEC-MoM are valid as long as the wire radius is much smaller than the wavelength and the wire length. The result is a set of coupled integral equations for the currents. As a practical matter, these approximations limit the application of NEC-MoM to resonance and below for a given structure.

The integral equations are solved numerically by the collocation form of the method of moments, which involves expanding the unknown current in a summation of basis functions and enforcing the equality of weighted integrals of the fields. This reduces the integral equations to a matrix equation (Harrington, 1968). The weighting functions are chosen to be delta functions. This collocation method results in a point sampling of the fields. Wires are divided into short segments, with a sample point at the center of each segment.

The matrix equation that results from the moment method is solved by LU decomposition (Press et al., 1986). This determines the currents on the wire segments. From the currents, all electromagnetic parameters of interest can be derived: impedance, coupling, and near and far fields.

2.3 NEED FOR VISUALIZATION

Computational electromagnetics can be thought of as a three-step process: problem definition, computation, and solution description. The drawback to using a computational electromagnetics code such as NEC-MoM is in the effort required at each of the three steps. Preparing the input model for NEC and evaluating the output can be an overwhelming task. Performing calculations can be exceedingly time consuming for all but the most simple structures.

To conserve computing resources, input models must be extensively validated before calculations are performed. Enormous quantities of output data are generated, including currents on all the wire segments, near-field contours, and far fields. One of the goals of this IED project was to demonstrate how advanced visualization techniques could be used to assist in input validation and the rapid interpretation of output data.

3.0 VISUALIZATION TECHNIQUES

During this IED project, decisions had to be made regarding how to effectively visualize the data of interest. In many cases, there were several options. It was often difficult to decide which display method would have the most utility for the final user. The products of this project are now being used in support of ship EM design projects. This application of these products will quantify utility.

3.1 COLOR CODING

One of the main problems encountered was in deciding how to assign color coding. Color coding is an art unto itself. It was often difficult to decide whether color coding should be done using a continuous range of colors or using a discrete set of colors.

There are three components to color. The RGB color model interprets these components as the three colors (red, green, blue) used by the CRT. By varying the amount of each color component, the full range of screen colors can be achieved. The SGI machine has 24–bit color, with 8 bits for each component. Each component is an integer value

between 0 and 255. In this manner, the console is capable of displaying over 16 million different colors. Of course, it is doubtful that the human eye can distinguish all these.

Humans do not perceive colors in the same manner that the console displays them. To humans, it is more intuitive to use the HSV color model (hue, saturation, and value). The HSV model is based on the intuitive appeal of the artist's tint, shade, and tone. Hue has a value between 0 and 360. Saturation and value range between 0 and 1. Algorithms exist for translating between the different color models (Foley et al., 1990).

At one point during this IED project, it was proposed that various data components could be encoded into the different components of the HSV color model. For example, for complex data, the magnitude could be encoded in the hue and the phase could be encoded in the value. This idea was found to be not feasible for two reasons. First, the user became overwhelmed by the amount of information contained in slight variations in displayed color. Second, the color printer was totally incapable of reproducing anything but major variations in color, so one could not obtain a hard-copy output of the visualization display.

For most data sets, a discrete color-key coding system was found to provide the most useful visualization of the data. Seven bins were chosen and the data were linearly assigned to the bins. The color-key assignments were changed several times. A final decision was made to use colors that gave the best contrast when printed out on the color printer. Phase is displayed as a continuous range of hue, with saturation and value set to 1.0.

3.2 COMPLEX DATA

In most cases, we found that trying to display both components (real and imaginary or magnitude and phase) of a complex data set in one image confused the user unnecessarily. A decision was made to allow the user to display complex data in side-by-side windows if both components needed to be viewed simultaneously.

3.3 VECTOR DATA

Current data and field data are vectors. A straightforward method of displaying a vector's orientation was never developed. The current vector was defined relative to the wire's direction. It was felt that the direction of the current did not have as much value as the magnitude and phase of the current.

4.0 APPROACH

For advanced visualization, decisions had to be made regarding hardware and software. The authors were fortunate to have access to an SGI 4D/320GTXB. This machine represents the most sophisticated visualization hardware currently available. Several visualization packages were available on the SGI machine, including PV-Wave, apE, Explorer, and the SGI Graphics Library. The available high- level languages included C and Fortran.

4.1 SGI 4D/320

The SGI 4D/320GTXB is one of the SGI POWER Series™ line of computers. It can be configured with up to eight CPUs; the one used for this project has two 33-MHz CPUs. It has the following CPU performance ratings (SGI, 1991): 59 MIPs (VAX Dhrystone MIPS), 20 MFLOPS (DP Linpack 1000x1000), 41 SPECmarks. The GTXB has 48 bits color and 24 bits Z buffer. Its graphics performance is rated as 400K vectors/sec, 150K triangles/sec, and 100K polygons/sec. The system used for this project is configured with 64-MB memory and a 19-inch console monitor.

4.2 SOFTWARE PACKAGES

Several graphics software packages were available on the SGI computer. Due to time and monetary constraints, no investigation was made of purchasing other packages. A decision was made to work with the available packages. The available packages were: PV-Wave, apE, Explorer, and the SGI GL.

PV-Wave is a very powerful visualization package. Unfortunately, it did not seem to run properly on the SGI; moreover, it was only suited to graphically displaying data sets. It had no 3D model rendering capabilities that would be needed for the proposed IED project.

apE is a software toolkit for visualization. It was developed by the Ohio Supercomputer Graphics Project starting in 1987. Scientific and engineering data can be processed and viewed as plots, color images, and three-dimensional objects. Unfortunately, it was cumbersome to use. 3D objects had to be described using the apE format. It was very difficult to translate from a NEC input file to apE. For this reason, apE was not used for this IED project.

Explorer was developed by SGI to fully use the extraordinary capabilities of the SGI visualization computers. It is similar in design to apE, but it is much less cumbersome to use. Explorer is a system for creating powerful visualization *maps*, each of which comprises a series of small software tools, called *modules*. A map is a collection of modules that carries out a series of related operations on a data set and produces a visual representation of the result. Explorer has a data conversion utility for moving data between Explorer and other data formats. A module builder allows custom modules to be created. Unfortunately, Explorer became available only near the end of the IED project year, so it was only used for a small portion of the project. There was also concern about future portability, since Explorer is only available for the SGI system.

GL is a set of graphics and utility routines that provides high- and low-level support for graphics. The routines can be called from either C code or Fortran code. Though quite primitive, GL allows one to access all the powerful visualization capabilities of the SGIs, including 3D drawing, Gouraud shading, device polling, double buffering, coordinate transformations, hidden surface removal (z-buffering), lighting, pick correlation, and texturing, depending on the hardware's capabilities. SGI provides an enormous number of demo programs that can be easily modified. GL provides a straightforward way to develop the capabilities required of this IED project.

4.3 NEC DATA SETS

The NEC input data set is created by a program called NEEDS (Li, Logan, and Rockway, 1988). The data set is an ASCII file written in an archaic format with each line representing a "card." The first two characters on each line refer to an alphabetic code that determines what the data on that line represent. For example, "CM" is a comment line and "GW" is a wire description line. The output from NEC is an enormous ASCII text file in a format that is very difficult to read from within another computer program. For that reason, a parser program is needed to extract the data of interest and put such data in a software-readable format. A preprocessing filtering program, *filter*, was written in Fortran to put the input and output data in a form that would allow easy display of the data of interest. Appendix A contains a source code listing of this program.

Filter prompts the user for the name of a NEC input file. The user must enter the full filename. Then the user is asked whether there is an associated currents file. If the answer is yes, the user is prompted to enter the full name of the currents file. Finally, the user is prompted for the name for the output file. This file must have the extension "*.geo*" (for *geometry*).

Filter expects the format of the currents file to be as follows: two lines followed by the data listed as "magnitude phase". The current data for each segment are on their own line. The last line in the file must be "-1.234 -1.234".

5.0 VISUALIZATION PROGRAM STRUCTURE

The visualization programs that were developed for this IED project were written in the ANSI C language using the SGI Graphics Library to access the visualization tools. The program to visualize the problem definition and solution description is named *view_nec*, and a source code listing of it is in Appendix B. The program evolved as a result of feedback from users. However, the basic structure using multiple windows, pull-down menus, and transformations was determined from the beginning.

5.1 WINDOWS

The beauty of working in an X-windows type environment, such as that on the SGI machines, is the flexibility it affords the user. That flexibility was retained in the development of *view_nec*. Each data component is displayed using 3D imagery in its own window. At any given moment during the running of the program, the user has complete control over how many data windows are displayed as well as their sizes and locations. In addition, multiple copies of *view_nec* can be launched to do side-by-side comparisons of different NEC runs.

5.2 MENUS

Pressing the right-hand mouse button while the mouse cursor is within any *view_nec* window brings up a "pull-down" menu. This pull-down menu allows the user to select

which data windows to open or close, which transformation is active, and whether a coordinate axis is displayed. Once a selection has been made, the affected windows are updated.

5.3 TRANSFORMATIONS

Entire books have been written on the display of 3D models. This section will just summarize the display capabilities available in *view_nec*. The available transformations include rotation, translation, and zooming. Each 3D transformation is represented internally in the computer by a 4 × 4 matrix. The IRIS Geometry Engines transform all geometric data (vertices of points, lines, and polygons) by multiplying each vertex by the accumulated matrices. In *view_nec,* the transformations are actuated with the left mouse button held down and the cursor dragged across the screen.

There are three basic classes of transformations that can be carried out by the Graphics Library: projection transformations, viewing transformations, and modeling transformations. A good analogy is to a camera with a versatile lens. Projection transformations describe the type of lens on the camera. Viewing transformations determine where the camera is positioned and in which direction it is pointed. Finally, modeling transformations affect the location, orientation, and size of the 3D geometric models in the scene.

There is often more than one way to carry out the transformation of a scene. For instance, instead of moving the camera toward the object, the object could be moved closer to the camera. However, there may be subtle differences. The transformation methods used in *view_nec* were chosen by selecting demo programs that gave the desired effect.

5.3.1 Zoom

The zoom feature allows the user to expand a selected portion of the model for a more detailed display.

Zooming is achieved internally by using a projection transformation, `perspective`. By modifying the `fovy` parameter, the field of view in the y dimension is modified. This has the effect of making the scene appear closer or farther away. The aspect ratio between the field of view in x and the field of view in y and the distances to the near and far clipping planes are kept constant.

5.3.2 Rotate

The rotate feature allows the user to rapidly change the orientation of the model. This permits a better feeling for the three-dimensional nature of the model and often allows various features of the data to become more apparent.

Rotation of the scene is carried out internally using the `polarview` command. `Polarview` is a viewing transformation. This command moves the viewpoint. The distance from the origin is kept constant while the azimuthal angle in the x–y plane and the

incidence angle in the y-z plane are modified. This is akin to moving the camera over a spherical surface surrounding the scene.

5.3.3 Translate

The translate feature allows the user to move the model up or down or left or right in the plane of the screen. This shifts the origin and is useful before zooming in on a selected portion of the model.

Translation of the scene is carried out internally by using the modeling transformation `translate`. The program is set up to allow only translation of the object in the plane of the screen. The size of the object is preserved.

5.4 PICK CORRELATION

It quickly became apparent that a method was needed to selectively choose a single element, such as a wire segment, from the 3D wire object display. This was needed for troubleshooting as well as linking the display back to the NEC input data set. The middle mouse button was chosen for this purpose. The technique used is known as *pick correlation*. The Graphics Library provides this capability.

Pick correlation identifies objects on the screen that appear near the mouse cursor. Information about these objects is stored in a buffer. *View_nec* uses this information to allow the user to select a single wire segment from the object displayed and list all information about it. A sample wire parameters window is shown in figure 2.

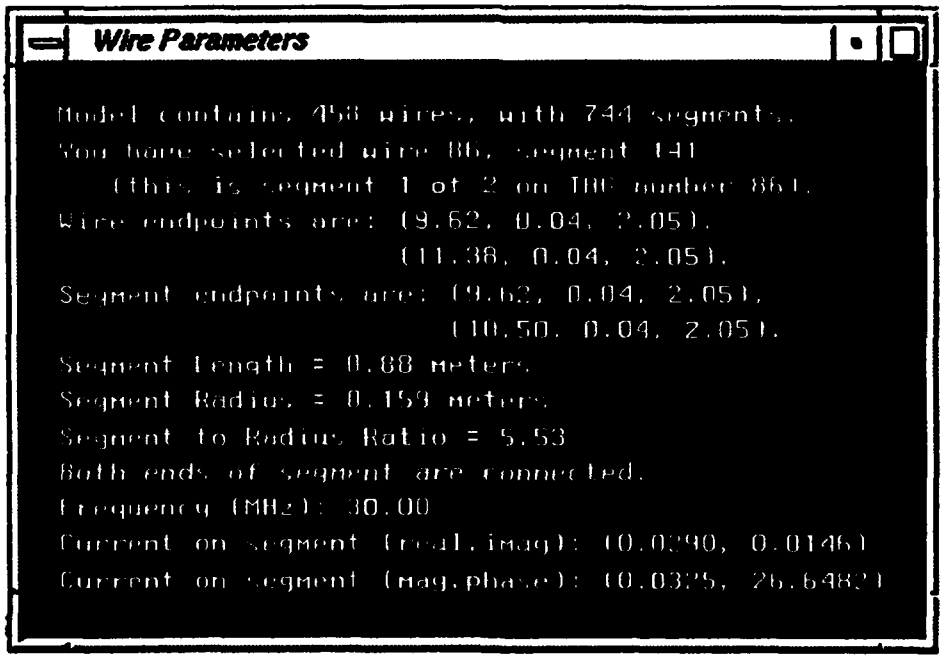


Figure 2. Wire parameters window.

5.5 USER'S GUIDE

The user does not need to understand the internal structure of *view_nec* that is described above in order to use the program. Running *view_nec* is a very straightforward procedure.

If the user is interested in viewing only the geometry description products or wire currents, then just the file with extension "*.geo*" is required. This file contains the geometry of the model and the currents on the wires if these have been calculated by an NEC run. The "*.geo*" file is generated by running the program *filter* as described above. If the user is interested in examining the near or far fields, then additional files are needed. The far-field data are contained in four files: **.etm* (magnitude of E-theta component), **.etp* (phase of E-theta component), **.epm* (magnitude of E-phi component), and **.epp* (phase of E-phi component). The near-field data are contained in the file **.fld*. The user must generate these files before running *view_nec*. For the required format for each of these files, consult the source code listing of *view_nec* in Appendix B.

To run *view_nec* type "*view_nec*". The program then prompts the user for the file name. This is the file name for the **.geo* file. DO NOT TYPE THE *.geo* EXTENSION! During the running of *view_nec*, the program will automatically access any additional files needed for the display of near or far fields if requested. All the files must have the same prefix. The extensions must be assigned as outlined above.

During the running of *view_nec*, the menu can be brought up at any time by pressing the right-hand mouse button while the mouse cursor is within any *view_nec* data window. The menu allows the user to open or close any data windows, change the transformation mode, or toggle the coordinate axis display.

The transformation mode is indicated in the lower left corner of all data windows. The transformation is carried out by clicking the left mouse button and dragging the cursor within any data window.

Clicking on the wire model with the middle mouse button brings up the wire parameter window described above. The currently selected wire segment flashes until it is unselected (by either clicking the middle mouse button in the background or selecting a new wire segment).

Clicking the middle mouse button in the near-fields window changes the near-field display threshold (this is described below in the solution description products section).

To exit *view_nec*, click the left mouse button in the shaded band at the very top of any display window and select "*close*".

6.0 GEOMETRY DESCRIPTION PRODUCTS

The NEC input file describes the modeled object (such as a ship) as a collection of wires. The description of each wire includes: a tag number (for identification purposes), the number of segments on the wire, the (x,y,z) coordinates for both the beginning and end points, and the radius of the wire. Since some wires are tapered, extra information is given for these wires to describe the tapering.

There are four windows that can be opened up to display the model's input geometry. The windows each display one of the following: Wire Segmentation (segment length in meters), Wire Radius (in meters), Segment to Radius Ratio, or Wire Connectivity (none, one, or both ends connected). All of these windows display the model as a 3D wire object. The data are encoded in the color of each wire segment using a linear color assignment scheme. A color key is displayed in the lower left corner of the window. Figure 3 is a sample window showing wire segmentation. Without color, one cannot get a true understanding of the "value" of the display to the user.

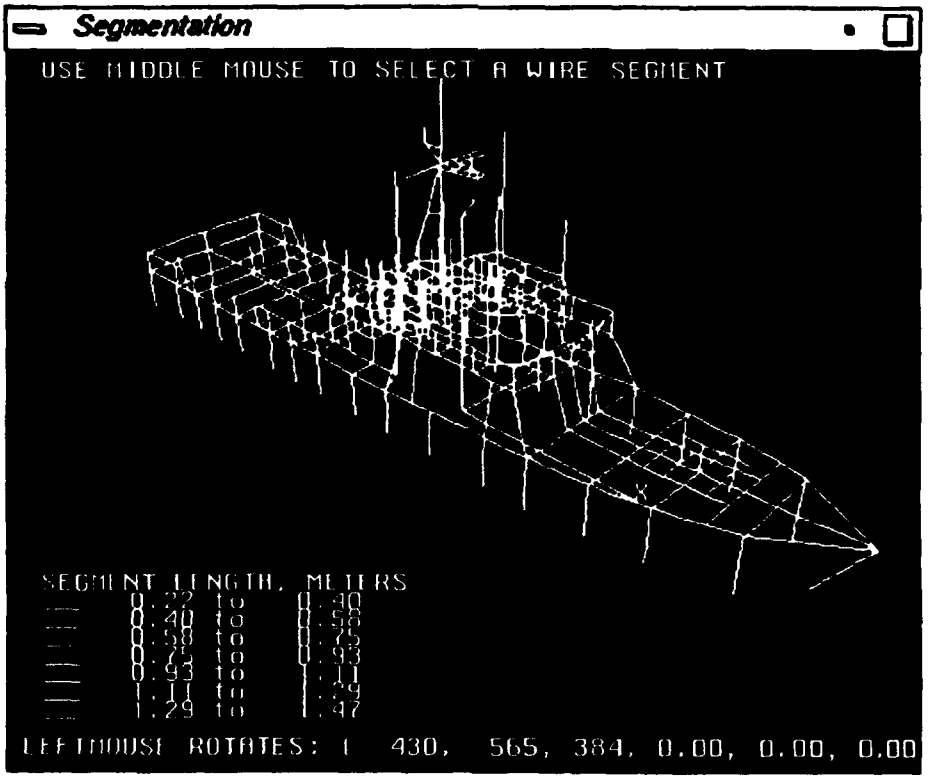


Figure 3. *view_nec* wire segmentation display window.

7.0 SOLUTION DESCRIPTION PRODUCTS

Solution description is displayed in up to eight windows. These eight windows include currents on the wires (real component, imaginary component, magnitude, or phase), total near field (electric field, in volts/meter), z-component of near field, theta component of far field, and phi component of far field.

7.1 CURRENTS

Currents on the wires are displayed using the same technique as was used for the geometry description products: current is color-coded on a 3D wire display of the model. Since currents are complex, there are four different current data windows that can be displayed: real component, imaginary component, magnitude, and phase. Most users have found the magnitude window to be most useful.

7.2 NEAR FIELDS

NEC-MoM allows the user to calculate the near field at selected locations surrounding the model. These locations are usually defined by a 3D grid surrounding the model. The near field is a complex vector. Of most interest is the z-component, since this will be the component that affects personnel standing on the surface of the ship. Several techniques were investigated for displaying the near-field data points.

View_nec displays near fields using a "fog" technique in which the density of activated pixels in the image is linearly proportional to the field intensity at the nearest calculation point. Since the calculation points usually form a 3D grid, the image contains square blocks of fog varying in density with the field intensity. The fog is color-coded in a manner similar to that used for the geometry description products. The 3D wire model is drawn using a dark gray color so as not to detract from the near-field display.

The near-field windows have a thresholding capability. By clicking in the window with the middle mouse button, the user is able to selectively change the threshold at which the fields are displayed. This allows the user to quickly determine what areas surrounding the model have fields above a particular cutoff level. The threshold can be selected to be any of the color-key bin levels, HERP (Hazardous Electromagnetic Radiation for Personnel), or HERO (Hazardous Electromagnetic Radiation for Ordnance). HERP and HERO are functions of the frequency at which the NEC model was run. The near-field windows display the value of the frequency. HERP and HERO values are shown in table 1.

Table 1. HERP and HERO near-field values.

Frequency Range, MHz	Maximum E-Field, V/m
HERP	
0.01–3	632.5
3–30	1897/(frequency)
30–300	63.25
300–1500	3.65 × SQRT (frequency)
1500–300,000	141.4
HERO	
0.1–1	100/(frequency)
1–3.7	$100/(\text{frequency}^3)$
3.7–10	2
10–1000	$0.5 \times (\text{frequency}^{0.6})$

7.3 FAR FIELDS

There are two windows available for the display of far fields. These windows allow the user to display either the theta component or the phi component of the far electric field.

The far field is complex data. This makes display of it somewhat complicated. A method was sought of displaying both the magnitude and phase simultaneously. The method selected involves displaying the fields as a three-dimensional surface. The distance from a point on the surface to the origin is proportional to the field magnitude at that point. The color at the point is determined by the phase of the field at that point. Gouraud shading is used between points to allow for color transition.

8.0 MoM MATRIX DISPLAY

There has been some speculation that visualizing the method of moments matrix that has been extracted from NEC might give some insight into the wire model's validity. The MoM matrix is a complex matrix that may have a dimension of about 1000. The dimension of the matrix is equal to the number of wire segments in the input model. This matrix is also referred to as the impedance matrix. Two methods were used to render the data: an Explorer map and a Graphics Library program.

8.1 EXPLORER MAP

An Explorer map was developed to display the magnitude of the matrix. This map reads in the data, subsamples and crops the data, extracts the magnitude values, and then renders the data as a three-dimensional surface. The vertical offset of a point on the surface (as well as the color of the point) is proportional to the magnitude at that point.

There are several advantages to using the Explorer map. Explorer allows a great amount of flexibility in assigning color mappings, manipulating the rendered 3D object, and modifying of the vertical offset.

There are also several serious disadvantages to using the Explorer map. Foremost is the array size limitation (64K), which does not allow a display of the entire matrix for a typical ship. A typical ship matrix can have as many as a million elements. The array can be cropped and subsampled, but a great deal of data are lost. This makes the rendered graphic almost useless. In addition, only the magnitude of the data can be displayed. The phase component of the data cannot be displayed. Finally, there is no way for the user to determine where in the matrix (i.e., row and column) different features in the display are located.

8.2 GRAPHICS LIBRARY PROGRAM

A Graphics Library program, *view_mom*, was developed to allow the display of almost any size method-of-moments matrix. Its structure is similar to that of *view_nec*, but it is quite a bit simpler. *View_mom* allows data to be displayed in three display modes: 2D (surface), 1D (wire), or 0D (points). The pull-down menu is used to select the display mode.

View_mom allows the user to select two different windows. The first window has the data displayed as a three-dimensional object, with the vertical offset and color proportional to the logarithm of the magnitude of the data. The second window is similar, except that it has the color proportional to the phase of the data. Each window allows the surface to be rotated, translated, and zoomed, just as in *view_nec*. The right-hand mouse button brings up the menu. The left mouse button performs the transformations. The middle mouse button performs a pick correlation that allows the user to select a point on the matrix. After a matrix point has been picked an information window appears in the lower left corner of the console screen. This information window describes the matrix and gives information about the selected point. Included in this information are the row and column number of the point. A source code listing of *view_mom* is in Appendix C. A sample MoM matrix display window is shown in figure 4.

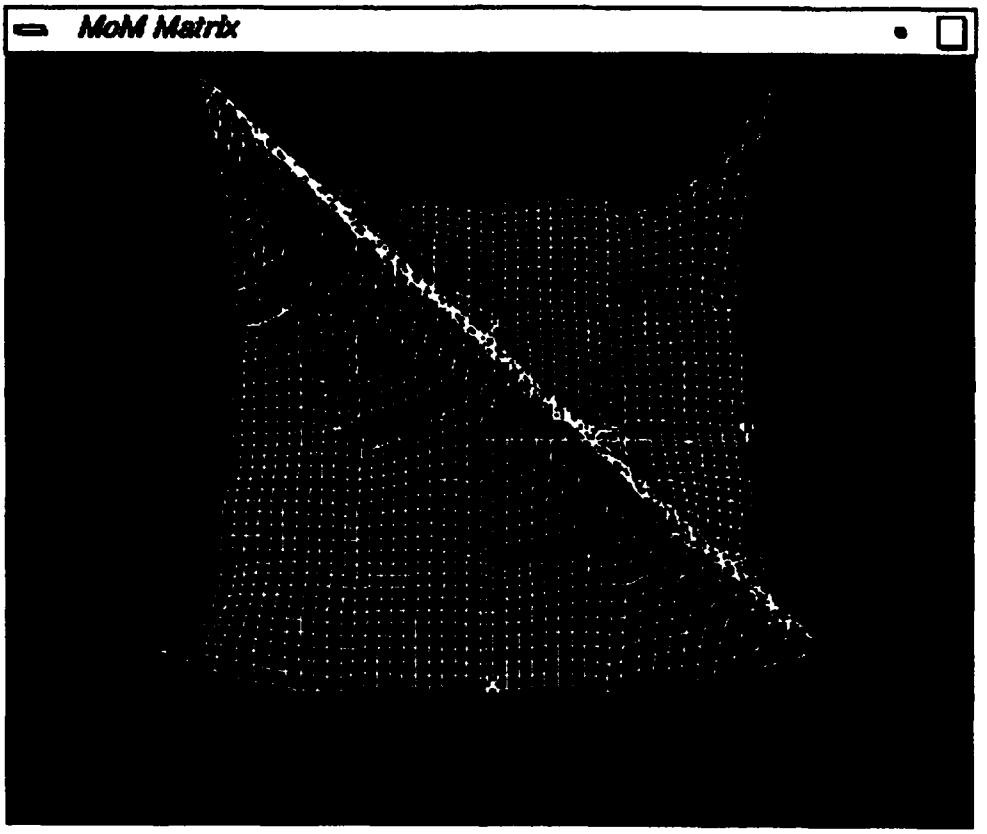


Figure 4. MoM matrix 2D display for a four-sided loop.

9.0 OBSERVATIONS

During this IED project the authors learned a great deal about visualization techniques and limitations. One observation is that as more and more features are added to a piece of software, the user interface becomes the limiting factor in the utility of the program. Another observation has to do with the limits of hard copy devices, such as the color printer that was attached to the SGI. Although it was a very high quality printer, it was

extremely limited in its ability to faithfully reproduce the hue, saturation, and brightness displayed on the console. Subtle differences in hue could not be discerned, and saturation variations were washed out.

9.1 VIDEO

In order to properly document this effort, the authors have proposed making a short (10-minute) video. This is the only way to demonstrate the utility of the codes that were developed. The video is still in production as of the date of this report.

9.2 PRODUCTION CODE

To be used as production codes, *view_nec* and *view_mom* need some additional work. The user interfaces need improvement. NASA's Goddard Space Flight Center has developed a transportable applications environment, TAE+, which provides a means of building a graphical user interface into applications. TAE+ is designed to run on the Silicon Graphics machines.

There needs to be a more robust method of extracting NEC output data for interface to a NEC visualization production code. It has been proposed that more CAD-like capabilities be added to *view_nec*.

9.3 PC IMPLEMENTATION

It is not feasible right now to port the visualization codes to the PC. Although they are written in the C–language, the Graphics Library is specific to the Silicon Graphics machine. However, SGI is attempting to make their GL a graphics standard that will be available in the future for PCs. In the meantime, SGI is introducing a line of desktop visualization machines, the Indigos, with excellent graphics and CPU performance at an exceptionally low price, competitive with high-end PCs.

9.4 OTHER CEM CODES

Future work will focus on bringing visualization capabilities to other computational codes used in electromagnetic ship design. These codes include the NEC–Basic Scattering Code developed by Ohio State University and TSAR, a finite difference time domain code developed by Lawrence Livermore National Laboratory.

10.0 CONCLUSIONS

Several payoffs have resulted, or will result, from the work done on this IED project.

- Visualization tools now exist for performing analysis and validation at all three steps of a NEC-MoM design procedure.
- These tools have greatly improved the utility of computational techniques used in electromagnetic analysis and are now being used to support topside ship design projects for the Navy.

- In combination with the work done during a related FY91 IED project, the visualization tools developed for this project are supporting the efficient transitioning of computational electromagnetics to high-performance computing.
- High-performance computing will provide a huge increase in processing speed and data analysis and permit the more accurate modeling of ship topsides.
- Since topside synthesis involves generating enormous quantities of data, design quality will significantly improve with the enhanced comprehension and computational speed offered by visualization and high-performance computing.
- This technology is in the process of being transitioned into the Electromagnetic Engineering (EME) project of NAVSEA.
- Based on the work done during this IED project and a related FY91 IED project, the 6.2 EMC project has scheduled projects in high-performance computing and visualization for FY94.

11.0 REFERENCES

Burke, G. J. 1989. "Recent Advances in NEC: Applications and Validation," UCRL–100651, Lawrence Livermore National Laboratory (March), Livermore, CA.

Burke, G. J., and A. J. Pogio. 1981. "Numerical Electromagnetic Code (NEC)-Method of Moments," NOSC Technical Document 116, Naval Ocean Systems Center, San Diego, CA.

Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. 1990. *Computer Graphics: Principles and Practice*, Second Edition, Addison-Wesley Publishing Company, NY.

Harrington, R. F. 1968. *Field Computation by Moment Methods*, The Macmillan Company, New York, NY.

Li, S. T., J. C. Logan, and J. W. Rockway. 1988. "Ship EM Design Technology," *Naval Engineers Journal*, May.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1986. *Numerical Recipes: the Art of Scientific Computing*, Cambridge University Press, New York, NY.

SGI, 1991. *Periodic Table of the IRISes*, Silicon Graphics Computer Systems.

Appendix A
SOURCE CODE FOR *filter*

```
      PROGRAM FILTER
C
      DIMENSION XWIRE1(3000),YWIRE1(3000),ZWIRE1(3000),
     $XWIRE2(3000),YWIRE2(3000),ZWIRE2(3000)
      DIMENSION XNODE1(3000),YNODE1(3000),ZNODE1(3000),ITAGS(3000),
     $XNODE2(3000),YNODE2(3000),ZNODE2(3000),SEGS(3000),RADS(3000),
     $SRRAT(3000),RCR(3000),PCR(3000)
      REAL ICR(3000),MCR(3000)
      LOGICAL CONECT(6000)
      REAL SEGMIN,SEGMAX,RADMIN,RADMAX,SRMIN,SRMAX,RCRMIN,RCRMAX,
     $ICRMIN,ICRMAX,MCRMIN,MCRMAX,XPCR,RDEL,BRAD,ERAD,LENGTH,
     $SLEN,SRAD
C
      INTEGER NSEGS,ITAG,IX,ISEG,IRAD,ISR,ICON,IRCR,IICR,IMCR
      INTEGER NWIRES,ISEGS(3000),JWIRE(3000),JSEG(3000)
      INTEGER NNODES,INODE,JNODE,I1,I2
      CHARACTER*12 INFIL
      CHARACTER*72 LINE
      CHARACTER*1 CH
C  get the filename
      WRITE(6,*) 'Enter NEC file name: '
      READ(5,10) INFIL
10    FORMAT(A)
C  open the NEC file
      OPEN(7, FILE=INFIL)
C  open the temporary output file
      OPEN(8, FILE='data.tmp')
      OPEN(9, FILE='freq.tmp')
C  loop through the NEC file reading the lines
20    READ(7,10,END=50) LINE
C  if its a wire line, write it out
      IF (LINE(1:2).EQ.'GW') WRITE(8,*) LINE(3:)
      IF (LINE(1:2).EQ.'GC') WRITE(8,*) LINE(3:)
      IF (LINE(1:2).EQ.'FR') WRITE(9,*) LINE(3:)
      GOTO 20
50    CLOSE(7)
      CLOSE(8)
      CLOSE(9)
C  open up temporary frequency file and read frequency
      OPEN(9, FILE='freq.tmp')
      FREQ = 299.8
      READ(9,*,END=60) ISTEP,ICNT,IA,IB,FREQ,XSTEP
60    CLOSE(9)
C  open up temporary file and read in wire lines
      OPEN(7, FILE='data.tmp')
C  initialize
      CDR=ACOS(0.0)/90.0
      NSEGS = 0
      NNODES = 0
      NWIRES = 0
C  loop, reading data, also find minimum x, y, and z
100   READ(7,*,END=200) ITAG,IX,X1,Y1,Z1,X2,Y2,Z2,X
      NWIRES = NWIRES + 1
      ITAGS(NWIRES) = ITAG
      ISEGS(NWIRES) = IX
      XWIRE1(NWIRES) = X1
      XWIRE2(NWIRES) = X2
      YWIRE1(NWIRES) = Y1
      YWIRE2(NWIRES) = Y2
      ZWIRE1(NWIRES) = Z1
      ZWIRE2(NWIRES) = Z2
      IF (NSEGS.EQ.0) THEN
        XMIN = X1
        YMIN = Y1
        ZMIN = Z1
        SEGMIN = 10000.0
        RADMIN = 10000.0
        SRMIN = 10000.0
        RCRMIN = 10000.0
        ICRMIN = 10000.0
        MCRMIN = 10000.0
        XMAX = X1
        YMAX = Y1
        ZMAX = Z1
        SEGMAX = 0.0
        RADMAX = 0.0
        SRMAX = 0.0
        RCRMAX = -10000.0
        ICRMAX = -10000.0
        MCRMAX = 0.0
      END IF
      IF (X1.LT.XMIN) XMIN=X1
      IF (Y1.LT.YMIN) YMIN=Y1
      IF (Z1.LT.ZMIN) ZMIN=Z1
      IF (X1.GT.XMAX) XMAX=X1
      IF (Y1.GT.YMAX) YMAX=Y1
      IF (Z1.GT.ZMAX) ZMAX=Z1
      IF (X2.LT.XMIN) XMIN=X2
      IF (Y2.LT.YMIN) YMIN=Y2
      IF (Z2.LT.ZMIN) ZMIN=Z2
```

```
      IF (X2.GT.XMAX) XMAX=X2
      IF (Y2.GT.YMAX) YMAX=Y2
      IF (Z2.GT.ZMAX) ZMAX=Z2
      IF (X.GE.0.000001) THEN
      DO 120 I=1,IX
        NSEGS = NSEGS+1
        NNODES = NNODES+2
        JWIRE(NSEGS) = NWIRES
        JSEG(NSEGS) = I
        XNODE1(NSEGS) = X1 + (I-1)*(X2-X1)/IX
        YNODE1(NSEGS) = Y1 + (I-1)*(Y2-Y1)/IX
        ZNODE1(NSEGS) = Z1 + (I-1)*(Z2-Z1)/IX
        XNODE2(NSEGS) = X1 + I*(X2-X1)/IX
        YNODE2(NSEGS) = Y1 + I*(Y2-Y1)/IX
        ZNODE2(NSEGS) = Z1 + I*(Z2-Z1)/IX
        SEGS(NSEGS) = SQRT((X1-X2)**2+(Y1-Y2)**2+(Z1-Z2)**2)/IX
        RADS(NSEGS) = X
        SRRAT(NSEGS) = SEGS(NSEGS)/RADS(NSEGS)
        CONECT(NNODES) = .FALSE.
        CONECT(NNODES-1) = .FALSE.
        IF (SEGS(NSEGS).LT.SEGMIN) SEGMIN=SEGS(NSEGS)
        IF (RADS(NSEGS).LT.RADMIN) RADMIN=RADS(NSEGS)
        IF (SRRAT(NSEGS).LT.SRMIN) SRMIN=SRRAT(NSEGS)
        IF (SEGS(NSEGS).GT.SEGMAX) SEGMAX=SEGS(NSEGS)
        IF (RADS(NSEGS).GT.RADMAX) RADMAX=RADS(NSEGS)
        IF (SRRAT(NSEGS).GT.SRMAX) SRMAX=SRRAT(NSEGS)
120   CONTINUE
      ELSE
        READ(7,*,END=200) I1,I2,RDEL,BRAD,ERAD
        DO 140 I=1,IX
          NSEGS = NSEGS+1
          NNODES = NNODES+2
          JWIRE(NSEGS) = NWIRES
          JSEG(NSEGS) = I
          IF (I.GT.1) THEN
            X1 = XNODE2(NSEGS-1)
            Y1 = YNODE2(NSEGS-1)
            Z1 = ZNODE2(NSEGS-1)
          END IF
          LENGTH = SQRT((X1-X2)**2+(Y1-Y2)**2+(Z1-Z2)**2)
          IF (I.EQ.1) THEN
            IF (ABS(1-RDEL).GT.0.0001) THEN
              SLEN = LENGTH*(1-RDEL)/(1-RDEL**IX)
            ELSE
              SLEN = LENGTH/IX
            END IF
          ELSE IF (I.EQ.IX) THEN
            SLEN = LENGTH
          ELSE
            SLEN = RDEL*SLEN
          END IF
          SRAD = BRAD + (I-1)*(ERAD-BRAD)/(IX-1)
          XNODE1(NSEGS) = X1
          YNODE1(NSEGS) = Y1
          ZNODE1(NSEGS) = Z1
          XNODE2(NSEGS) = X1+(SLEN/LENGTH)*(X2-X1)
          YNODE2(NSEGS) = Y1+(SLEN/LENGTH)*(Y2-Y1)
          ZNODE2(NSEGS) = Z1+(SLEN/LENGTH)*(Z2-Z1)
          SEGS(NSEGS) = SLEN
          RADS(NSEGS) = SRAD
          SRRAT(NSEGS) = SLEN/SRAD
          CONECT(NNODES) = .FALSE.
          CONECT(NNODES-1) = .FALSE.
          IF (SEGS(NSEGS).LT.SEGMIN) SEGMIN=SEGS(NSEGS)
          IF (RADS(NSEGS).LT.RADMIN) RADMIN=RADS(NSEGS)
          IF (SRRAT(NSEGS).LT.SRMIN) SRMIN=SRRAT(NSEGS)
          IF (SEGS(NSEGS).GT.SEGMAX) SEGMAX=SEGS(NSEGS)
          IF (RADS(NSEGS).GT.RADMAX) RADMAX=RADS(NSEGS)
          IF (SRRAT(NSEGS).GT.SRMAX) SRMAX=SRRAT(NSEGS)
140     CONTINUE
      END IF
      GOTO 100
200   CLOSE(7)
C  see which segments are connected
      DO 300 I=1,NSEGS
        INODE = (I*2)-1
        IF (ZNODE1(I).LE.RADS(I)) CONECT(INODE)=.TRUE.
        IF (ZNODE2(I).LE.RADS(I)) CONECT(INODE+1)=.TRUE.
        DO 350 J=1,NSEGS
          IF (J.NE.I) THEN
            JNODE=(J*2)-1
            IF (.NOT.CONECT(INODE)) THEN
              DIST = SQRT((ZNODE1(J)-ZNODE1(I))**2+
     $           (YNODE1(J)-YNODE1(I))**2+(XNODE1(J)-XNODE1(I))**2)
              IF (DIST.LE.RADS(I)) THEN
                CONECT(INODE) = .TRUE.
                CONECT(JNODE) = .TRUE.
              END IF
              DIST = SQRT((ZNODE2(J)-ZNODE1(I))**2+
     $           (YNODE2(J)-YNODE1(I))**2+(XNODE2(J)-XNODE1(I))**2)
              IF (DIST.LE.RADS(I)) THEN
                CONECT(INODE) = .TRUE.
                CONECT(JNODE+1) = .TRUE.
```

```
            END IF
          END IF
          IF (.NOT.CONECT(INODE+1)) THEN
            DIST = SQRT((ZNODE1(J)-ZNODE2(I))**2+
     $         (YNODE1(J)-YNODE2(I))**2+(XNODE1(J)-XNODE2(I))**2)
            IF (DIST.LE.RADS(I)) THEN
              CONECT(INODE+1) = .TRUE.
              CONECT(JNODE) = .TRUE.
            END IF
            DIST = SQRT((ZNODE2(J)-ZNODE2(I))**2+
     $         (YNODE2(J)-YNODE2(I))**2+(XNODF2(J)-XNODE2(I))**2)
            IF (DIST.LE.RADS(I)) THEN
              CONECT(INODE+1) = .TRUE.
              CONECT(JNODE+1) = .TRUE.
            END IF
          END IF
          IF (CONECT(INODE).AND.CONECT(INODE+1)) GOTO 300
        END IF
 350    CONTINUE
 300  CONTINUE
C  calculate shifts
      XSHIFT = (XMAX + XMIN)/2.0
      YSHIFT = (YMAX + YMIN)/2.0
      ZSHIFT = (ZMIN)
C  calculate scale factor
      XDIF = XMAX-XMIN
      YDIF = YMAX-YMIN
      ZDIF = ZMAX-ZMIN
      DIFMAX = XDIF
      IF (YDIF.GT.DIFMAX) DIFMAX=YDIF
      IF (ZDIF.GT.DIFMAX) DIFMAX=ZDIF
      SCALE = 4.0/DIFMAX
C  get currents if necessary
C  see if there is an output currents file
      WRITE(6,*) 'Is there an output currents file? (Y:Yes, N:No):'
      READ(5,10) CH
      IF (CH.EQ.'N'.OR.CH.EQ.'n') THEN
        DO 320 I=1,NSEGS
          RCR(I) = 0.0
          ICR(I) = 0.0
          MCR(I) = 0.0
          PCR(I) = 0.0
 320    CONTINUE
        RCRMIN = 0.0
        RCRMAX = 0.0
        ICRMIN = 0.0
        ICRMAX = 0.0
        MCRMIN = 0.0
        MCRMAX = 0.0
      ELSE
C get the filename
        WRITE(6,*) 'Enter output currents filename: '
        READ(5,10) INFIL
C open the currents file
        OPEN(7,FILE=INFIL)
C loop through reading currents (magnitude, phase)
        READ(7,10) LINE
        READ(7,10) LINE
        DO 340 I=1,NSEGS
          READ(7,*) MCR(I),PCR(I)
          RCR(I) = MCR(I)*COS(CDR*PCR(I))
          ICR(I) = MCR(I)*SIN(CDR*PCR(I))
          IF (MCR(I).LT.MCRMIN) MCRMIN=MCR(I)
          IF (MCR(I).GT.MCRMAX) MCRMAX=MCR(I)
          IF (RCR(I).LT.RCRMIN) RCRMIN=RCR(I)
          IF (RCR(I).GT.RCRMAX) RCRMAX=RCR(I)
          IF (ICR(I).LT.ICRMIN) ICRMIN=ICR(I)
          IF (ICR(I).GT.ICRMAX) ICRMAX=ICR(I)
 340    CONTINUE
        CLOSE(7)
      END IF
C  now print it all out
C  get the filename
      WRITE(6,*) 'Enter output file name: '
      READ(5,10) INFIL
C  open the output file
      OPEN(8, FILE=INFIL)
      WRITE(8,*) FREQ
      WRITE(8,*) NWIRES, NSEGS
      WRITE(8,*) XMIN, XMAX
      WRITE(8,*) YMIN, YMAX
      WRITE(8,*) ZMIN, ZMAX
      WRITE(8,*) XSHIFT, YSHIFT
      WRITE(8,*) ZSHIFT, SCALE
      WRITE(8,*) SEGMIN, SEGMAX
      WRITE(8,*) RADMIN, RADMAX
      WRITE(8,*) SRMIN, SRMAX
      WRITE(8,*) RCRMIN, RCRMAX
      WRITE(8,*) ICRMIN, ICRMAX
      WRITE(8,*) MCRMIN, MCRMAX
      DO 360 I=1,NWIRES
        WRITE(8,902) ITAGS(I),ISEGS(I),XWIRE1(I),YWIRE1(I),ZWIRE1(I),
     $XWIRE2(I),YWIRE2(I),ZWIRE2(I)
```

```
360   CONTINUE
902   FORMAT(' ',I5,I4,6F10.4)
      DO 400 I=1,NSEGS
        INODE=(I*2)-1
        IF (SEGMAX.EQ.SEGMIN) THEN
          ISEG = 0
        ELSE
          ISEG = (SEGS(I)-SEGMIN)/(SEGMAX-SEGMIN)*7
          IF (SEGS(I).EQ.SEGMAX) ISEG = 6
        END IF
        IF (RADMAX.EQ.RADMIN) THEN
          IRAD = 0
        ELSE
          IRAD = (RADS(I)-RADMIN)/(RADMAX-RADMIN)*7
          IF (RADS(I).EQ.RADMAX) IRAD = 6
        END IF
        IF (SRMAX.EQ.SRMIN) THEN
          ISR = 0
        ELSE
          ISR = (SRRAT(I)-SRMIN)/(SRMAX-SRMIN)*7
          IF (SRRAT(I).EQ.SRMAX) ISR = 6
        END IF
        IF (RCRMAX.EQ.RCRMIN) THEN
          IRCR = 0
        ELSE
          IRCR = (RCR(I)-RCRMIN)/(RCRMAX-RCRMIN)*7
          IF (RCR(I).EQ.RCRMAX) IRCR = 6
        END IF
        IF (ICRMAX.EQ.ICRMIN) THEN
          IICR = 0
        ELSE
          IICR = (ICR(I)-ICRMIN)/(ICRMAX-ICRMIN)*7
          IF (ICR(I).EQ.ICRMAX) IICR = 6
        END IF
        IF (MCRMAX.EQ.MCRMIN) THEN
          IMCR = 0
        ELSE
          IMCR = (MCR(I)-MCRMIN)/(MCRMAX-MCRMIN)*7
          IF (MCR(I).EQ.MCRMAX) IMCR = 6
        END IF
C put XPCR between 0 to 360
        XPCR = PCR(I)
 410    IF (XPCR.LT.0) THEN
          XPCR=XPCR+360.0
          GOTO 410
        END IF
 411    IF (XPCR.GE.360.0) THEN
          XPCR=XPCR-360.0
          GOTO 411
        END IF
        IF (CONECT(INODE).AND.CONECT(INODE+1)) ICON = 0
        IF (CONECT(INODE).AND.(.NOT.CONECT(INODE+1))) ICON = 2
        IF ((.NOT.CONECT(INODE)).AND.CONECT(INODE+1)) ICON = 2
        IF ((.NOT.CONECT(INODE)).AND.(.NOT.CONECT(INODE+1))) ICON = 4
        WRITE(8,903) JSEG(I),JWIRE(I),ISEG,IRAD,ISR,
     $ ICON,IRCR,IICR,IMCR,XPCR
 903    FORMAT(' ',I5,I5,7I3,F10.4)
       WRITE(8,905) SEGS(I),RADS(I),SRRAT(I),RCR(I),ICR(I),MCR(I)
 905   FORMAT(' ',6F10.4)
       WRITE(8,904) (XNODE1(I)),(YNODE1(I)),
      $(ZNODE1(I)),(XNODE2(I)),
      $(YNODE2(I)),(ZNODE2(I))
 904    FORMAT(' ',6F10.4)
400   CONTINUE
      CLOSE(8)
      END
```

Appendix B
SOURCE CODE FOR *view_nec*

```
 *                                  view_nec.c
 *       view_nec displays a nec wire model.  The polarview()
 *       viewing transformation is changed with mouse input when
 *       the left mouse button is held down.  Wire parameters are
 *     selected using the middle mouse button.  Menus are brought
 *     up using the right mouse button.
 *
 *       Copyright 1991, Silicon Graphics, Inc.  All Rights Reserved.
 *       See /usr/copyright for complete rights and liability information.
 *       Author:  Technical Education Course Developers
 *

#include <gl/gl.h>
#include <gl/device.h>
#include <stdio.h>
#include <math.h>
#define MAXPICK 50


 *       function prototypes
 *

void initialize(void);
void main(void);
void printhits(long iblink);
void hsv2rgb(float h);
void drawscene(short dxr, short dxt, short dxz,
               short dyr, short dyt, short dyz,
               long data_choice, long trans_choice, long iblink,
               short buffer[], long hits, Boolean picking, long do_win);
void axes(void);
void initethiv(void);
void initeph(void);
void initfld(void);
void initmodel(void);
void wmodel(long data_choice, long iblink);

FILE *geofile;
FILE *etmfile;
FILE *etpfile;
FILE *epmfile;
FILE *eppfile;
FILE *fldfile;
char file_name[25];
char geo_file[25];
char etm_file[25];
char etp_file[25];
char epm_file[25];
char epp_file[25];
char fld_file[25];
static char geo[] = ".geo";
static char etm[] = ".etm";
static char etp[] = ".etp";
static char epm[] = ".epm";
static char epp[] = ".epp";
static char fld[] = ".fld";
int zthresh = 0;
int fthresh = 0;
int nwires, nsegs, iseqs[3000], isw[3000], jwire[3000], data[3000][7];
int itags[3000];
static Boolean draw_axes = TRUE;
float bgnnode[3000][3], endnode[3000][3], bgnwire[3000][3], endwire[3000][3];
float mxmin, mxmax, mymin, mymax, mzmin, mzmax, mscale, xdata[3000];
float xshift, yshift, zshift, freq, herp, hero;
float segs[3000],rads[3000],srrat[3000],rcr[3000],icr[3000],mcr[3000];
float segmin, segmax, radmin, radmax, srmin, srmax, rcrmin, rcrmax,
      ircmin, icrmax, mcrmin, mcrmax, ethmax, ephmax, fldmax, zfldmax;
long rgbcol[3];

static float tx, ty, tz;
/* window id's */
long seg_win, rad_win, rat_win, con_win, real_win, imag_win,
     mag_win, ph_win, eth_win, eph_win, data_win, cur_win,
     fld_win, zfld_win;
static char seg_win_name[] = "Segmentation";
static char rad_win_name[] = "Wire radius";
static char rat_win_name[] = "Segment to radius";
static char con_win_name[] = "Connections";
static char real_win_name[] = "Real current";
static char imag_win_name[] = "Imaginary current";
static char mag_win_name[] = "Magnitude of current";
static char ph_win_name[] = "Phase of current";
static char eth_win_name[] = "E-theta";
static char eph_win_name[] = "E-phi";
static char data_win_name[] = "Wire Parameters";
static char fld_win_name[] = "Near field";
static char zfld_win_name[] = "Z-comp of Near field";
long xmax, ymax, zval;
```

```
/*                  initialize
 *        The initialize subroutine positions the window and specifies
 *        its future constraints.  Graphics configurations are set and
 *        the event queue is initialized.
 */

void
initialize(void)
{

        printf(" Enter file name: ");
        scanf("%s", file_name);
        strcpy(geo_file,file_name);
        strcpy(etm_file,file_name);
        strcpy(etp_file,file_name);
        strcpy(epm_file,file_name);
        strcpy(epp_file,file_name);
        strcpy(fld_file,file_name);
        strcat(geo_file,geo);
        strcat(etm_file,etm);
        strcat(etp_file,etp);
        strcat(epm_file,epm);
        strcat(epp_file,epp);
        strcat(fld_file,fld);
        if((geofile = fopen(geo_file,"r")) == (FILE*)0)
        {
          printf("\n\t\t File %s could not be opened. \n", geo_file);
          exit();
        }
          xmax = getgdesc(GD_XPMAX);
          ymax = getgdesc(GD_YPMAX);
        zval = getgdesc(GD_ZMAX);

          minsize(xmax/10, ymax/10);
        maxsize(xmax-200, ymax-160);
          keepaspect(xmax, ymax);
          seg_win = winopen("");
        wintitle(seg_win_name);
          winconstraints();
        winset(seg_win);
        zbuffer(TRUE);
          doublebuffer();
          RGBmode();
          gconfig();

          shademodel(FLAT);
          mmode(MVIEWING);

          qdevice(LEFTMOUSE);
          qdevice(MIDDLEMOUSE);
        qdevice(RIGHTMOUSE);
          qdevice(ESCKEY);
          tie(LEFTMOUSE, MOUSEX, MOUSEY);


}         /*   end initialize()   */


/*                  main
 *        main calls initialize, then goes into a loop.  Within the loop
 *        drawscene is called and events in the event queue are processed
 *        until user exits (by pressing ESCAPE or through the window manager).
 *        Mouse input is passed to drawscene when the left mouse button
 *        is held down.
 */

void
main(void)
{
          Boolean exitflag = FALSE;
          Boolean redraw_needed = TRUE;
        Boolean draw_seg = TRUE;
        Boolean draw_rad = FALSE;
        Boolean draw_rat = FALSE;
        Boolean draw_con = FALSE;
        Boolean draw_real = FALSE;
        Boolean draw_imag = FALSE;
        Boolean draw_mag = FALSE;
        Boolean draw_ph = FALSE;
        Boolean draw_eth = FALSE;
        Boolean draw_eph = FALSE;
        Boolean draw_data = FALSE;
        Boolean draw_fld = FALSE;
        Boolean draw_zfld = FALSE;
          short value;
        short buffer[BUFSIZE];
        long hits = 0;
        Boolean picking = FALSE;
        short deltax, deltay;
          long dev;
        long choice;
        long data_choice, trans_choice, iblink;
```

```
long menu, datamenu, transmenu, geomenu, curmenu,
     nfieldmenu, ffieldmenu;
  short x, oldx;            /*  mouse movement */
  short y, oldy;
  short dxr, dxt, dxz = 0;
  short dyr, dyt, dyz = 0;
static long whitecol[] = { 255, 255, 255 };
static long bluecol[] = { 0, 0, 255 };
static long dbluecol[] = { 0, 0, 128 };

  initialize();
data_choice = 1;
trans_choice = 1;
iblink = 0;
tx = 0;
ty = 0;
tz = 0;

/* make the geometry menu */
geomenu = defpup("Choose Geometry Data Window %t|"
  "Segmentation %x1|Wire radius %x2|Segment to radius ratio %x3|"
  "Wire connections %x4");
/* make the currents menu */
curmenu = defpup("Choose Currents Data Window %t|"
  "Real component of current %x5|"
  "Imaginary component of current %x6|Magnitude of current %x7|"
  "Phase of current %x8");
/* make the far fields menu */
ffieldmenu = defpup("Choose Far Fields Data Window %t|"
  "E-theta component %x9|E-phi component %x10");
/* make the near fields menu */
nfieldmenu = defpup("Choose Near Fields Data Window %t|"
  "Z-component of E-normal %x11|Total E-normal %x12");
/* make the data menu */
datamenu = defpup("Choose Data Window to Open/Close %t|"
  "Geometry %m|Currents %m|Near Fields %m|Far Fields %m",
  geomenu, curmenu, nfieldmenu, ffieldmenu);
/* make the transformation menu */
transmenu = defpup("Choose Transformation %t|"
  "Rotate %x13|Translate %x14|Zoom %x15|Return to initial state %x16");
/* make the main menu */
menu = defpup("Menu %t|Data Window %m|Transformation %m|"
  "Close Current Window %x17|Toggle Axes On/Off %x18",datamenu,transmenu);

  while (exitflag == FALSE)
  {
            if (redraw_needed == TRUE)
            {
        if (draw_seg)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    1, trans_choice, iblink,
                    buffer, hits, picking, seg_win);
        if (draw_rad)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    2, trans_choice, iblink,
                    buffer, hits, picking, rad_win);
        if (draw_rat)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    3, trans_choice, iblink,
                    buffer, hits, picking, rat_win);
        if (draw_con)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    4, trans_choice, iblink,
                    buffer, hits, picking, con_win);
        if (draw_real)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    5, trans_choice, iblink,
                    buffer, hits, picking, real_win);
        if (draw_imag)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    6, trans_choice, iblink,
                    buffer, hits, picking, imag_win);
        if (draw_mag)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    7, trans_choice, iblink,
                    buffer, hits, picking, mag_win);
        if (draw_ph)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    8, trans_choice, iblink,
                    buffer, hits, picking, ph_win);
        if (draw_eth)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    9, trans_choice, iblink,
                    buffer, hits, picking, eth_win);
        if (draw_eph)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    10, trans_choice, iblink,
                    buffer, hits, picking, eph_win);
        if (draw_zfld)
            drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
                    11, trans_choice, iblink,
                    buffer, hits, picking, zfld_win);
        if (draw_fld)
```

```
drawscene(dxr, dxt, dxz, dyr, dyt, dyz,
        12, trans_choice, iblink,
        buffer, hits, picking, fld_win);

          if (iblink == 0) redraw_needed = FALSE;
    iblink = -iblink;
}

while ( (exitflag == FALSE) &&
          (qtest() || (redraw_needed == FALSE)) )
{
          dev = qread (&value);
          if (dev == ESCKEY)
          {
                    if (value == 0)

                              exitflag = TRUE;
          }
          else if (dev == REDRAW)
          {
                    reshapeviewport();
                    redraw_needed = TRUE;
          }
    else if (dev == INPUTCHANGE)
    {
            cur_win = value;
    }
    else if (dev == RIGHTMOUSE && value == 1)
    {
            choice = dopup(menu);
            if (choice >= 1)
            {
            if (choice == 1)
            {
              draw_seg = !draw_seg;
              if (draw_seg)
              {
                minsize(xmax/10,ymax/10);
                maxsize(xmax-200,ymax-160);
                keepaspect(xmax,ymax);
                seg_win = winopen("");
                wintitle(seg_win_name);
                winconstraints();
                winset(seg_win);
                zbuffer(TRUE);
                doublebuffer();
                RGBmode();
                gconfig();
                shademodel(FLAT);
                mmode(MVIEWING);
              }
              else winclose(seg_win);
            }
            if (choice == 2)
            {
              draw_rad = !draw_rad;
              if (draw_rad)
              {
                minsize(xmax/10,ymax/10);
                maxsize(xmax-200,ymax-160);
                keepaspect(xmax,ymax);
                rad_win = winopen("");
                wintitle(rad_win_name);
                winconstraints();
                winset(rad_win);
                zbuffer(TRUE);
                doublebuffer();
                RGBmode();
                gconfig();
                shademodel(FLAT);
                mmode(MVIEWING);
              }
              else winclose(rad_win);
            }
            if (choice == 3)
            {
              draw_rat = !draw_rat;
              if (draw_rat)
              {
                minsize(xmax/10,ymax/10);
                maxsize(xmax-200,ymax-160);
                keepaspect(xmax,ymax);
                rat_win = winopen("");
                wintitle(rat_win_name);
                winconstraints();
                winset(rat_win);
                zbuffer(TRUE);
                doublebuffer();
                RGBmode();
                gconfig();
                shademodel(FLAT);
                mmode(MVIEWING);
              }
```

```
  else winclose(rat_win);
}
if (choice == 4)
{
  draw_con = !draw_con;
  if (draw_con)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    con_win = winopen("");
    wintitle(con_win_name);
    winconstraints();
    winset(con_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(con_win);
}
if (choice == 5)
{
  draw_real = !draw_real;
  if (draw_real)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    real_win = winopen("");
    wintitle(real_win_name);
    winconstraints();
    winset(real_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(real_win);
}
if (choice == 6)
{
  draw_imag = !draw_imag;
  if (draw_imag)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    imag_win = winopen("");
    wintitle(imag_win_name);
    winconstraints();
    winset(imag_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(imag_win);
}
if (choice == 7)
{
  draw_mag = !draw_mag;
  if (draw_mag)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    mag_win = winopen("");
    wintitle(mag_win_name);
    winconstraints();
    winset(mag_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(mag_win);
}
if (choice == 8)
{
  draw_ph = !draw_ph;
  if (draw_ph)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
```

```
    keepaspect(xmax,ymax);
    ph_win = winopen("");
    wintitle(ph_win_name);
    winconstraints();
    winset(ph_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(ph_win);
}
if (choice == 9)
{
  draw_eth = !draw_eth;
  if (draw_eth)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    eth_win = winopen("");
    wintitle(eth_win_name);
    winconstraints();
    winset(eth_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(GOURAUD);
    mmode(MVIEWING);
  }
  else winclose(eth_win);
}
if (choice == 10)
{
  draw_eph = !draw_eph;
  if (draw_eph)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    eph_win = winopen("");
    wintitle(eph_win_name);
    winconstraints();
    winset(eph_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(GOURAUD);
    mmode(MVIEWING);
  }
  else winclose(eph_win);
}
if (choice == 11)
{
  draw_zfld = !draw_zfld;
  if (draw_zfld)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    zfld_win = winopen("");
    wintitle(zfld_win_name);
    winconstraints();
    winset(zfld_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
    shademodel(FLAT);
    mmode(MVIEWING);
  }
  else winclose(zfld_win);
}
if (choice == 12)
{
  draw_fld = !draw_fld;
  if (draw_fld)
  {
    minsize(xmax/10,ymax/10);
    maxsize(xmax-200,ymax-160);
    keepaspect(xmax,ymax);
    fld_win = winopen("");
    wintitle(fld_win_name);
    winconstraints();
    winset(fld_win);
    zbuffer(TRUE);
    doublebuffer();
    RGBmode();
    gconfig();
```

```
            shademodel(FLAT);
            mmode(MVIEWING);
          }
          else winclose(fld_win);
        }
        if ((choice >= 13)&&(choice <= 16))
           trans_choice = choice - 12;
        if (trans_choice == 4)
        {
          dxr = 0;
          dxt = 0;
          dxz = 0;
          dyr = 0;
          dyt = 0;
          dyz = 0;
          tx = 0;
          ty = 0;
          tz = 0;
          trans_choice = 1;
        }
        if (choice == 17)
        {
          if (cur_win == seg_win) draw_seg = !draw_seg;
          if (cur_win == rad_win) draw_rad = !draw_rad;
          if (cur_win == rat_win) draw_rat = !draw_rat;
          if (cur_win == con_win) draw_con = !draw_con;
          if (cur_win == real_win)draw_real=!draw_real;
          if (cur_win == imag_win)draw_imag=!draw_imag;
          if (cur_win == mag_win) draw_mag = !draw_mag;
          if (cur_win == ph_win) draw_ph = !draw_ph;
          if (cur_win == eth_win) draw_eth = !draw_eth;
          if (cur_win == eph_win) draw_eph = !draw_eph;
          if (cur_win == zfld_win)draw_zfld=!draw_zfld;
          if (cur_win == fld_win) draw_fld = !draw_fld;
          winclose(cur_win);
        }
        if (choice == 18) draw_axes = !draw_axes;
        redraw_needed = TRUE;
        }
}
      else if (dev == LEFTMOUSE)
      {
                if (value)
                /* read initial mouse x,y, which will be
                 * next 2 events because tied to leftmouse
                 */
                {
                          qread(&x);
                          qread(&y);
                        /* now listen to all mouse x,y
                         * as long as mouse down */
                          qdevice(MOUSEX);
                          qdevice(MOUSEY);
                }
                else
                /* as soon as mouse released,
                 * stop listening to mouse x,y */
                {
                          unqdevice(MOUSEX);
                          unqdevice(MOUSEY);
                }
      }
else if (dev == MIDDLEMOUSE && value == 0)
{
        /* do a pick correlation in current window */
        winset(cur_win);
        data_choice = 0;
        if (cur_win == seg_win) data_choice = 1;
        if (cur_win == rad_win) data_choice = 2;
        if (cur_win == rat_win) data_choice = 3;
        if (cur_win == con_win) data_choice = 4;
        if (cur_win == real_win) data_choice = 5;
        if (cur_win == imag_win) data_choice = 6;
        if (cur_win == mag_win) data_choice = 7;
        if (cur_win == ph_win) data_choice = 8;
        if (cur_win == eth_win) data_choice = 9;
        if (cur_win == eph_win) data_choice = 10;
        if (cur_win == zfld_win) data_choice = 11;
        if (cur_win == fld_win) data_choice = 12;
        if (data_choice == 11)
        {
          zthresh++;
          if (zthresh == 9) zthresh = 0;
        }
        else if (data_choice == 12)
        {
          tthresh++;
          if (tthresh == 9) tthresh = 0;
        }
        else if ((data_choice >= 1)&&(data_choice <= 8))
        {
        pick(buffer,BUFSIZE);
        picking = TRUE;
```

```
                        deltax = 5;
                        deltay = 5;
                        picksize(deltax, deltay);
                        iblink = 0;
                        if (data_choice > 0)
                        {
                        drawscene(dxr,dxt,dxz,dyr,dyt,dyz,
                                  data_choice, trans_choice, iblink,
                                  buffer, hits, picking, cur_win);
                        }
                        picking = FALSE;
                        hits = endpick(buffer);
                        if (hits > 0)
                        {
                          iblink = buffer[1];
                          if (draw_data == FALSE)
                          {
                            prefposition(xmax/40,xmax*8/20,ymax/40,
                                         ymax*13/40);
                            data_win = winopen("");
                            wintitle(data_win_name);
                            winconstraints();
                            winset(data_win);
                            singlebuffer();
                            RGBmode();
                            gconfig();
                            shademodel(FLAT);
                            mmode(MVIEWING);
                            draw_data = TRUE;
                          }
                          winset(data_win);
                          c3i(dbluecol);
                          clear();
                          c3i(whitecol);
                          ortho2(0.0, 5.0, 0.0, 4.0);
                          printhits(iblink);
                        }
                        else
                        {
                          if (draw_data)
                          {
                            draw_data = FALSE;
                            winclose(data_win);
                          }
                        }
                        }
                        redraw_needed = TRUE;
                    }
                        else if (dev == MOUSEX)
                        {
                                  oldx = x;
                                  x = value;
                          if (trans_choice == 1) dxr = dxr + (x-oldx);
                          if (trans_choice == 2) dxt = dxt + (x-oldx);
                          if (trans_choice == 3) dxz = dxz + (x-oldx);
                                  redraw_needed = TRUE;
                        }
                        else if (dev == MOUSEY)
                        {
                                  oldy = y;
                                  y = value;
                          if (trans_choice == 1) dyr = dyr + (y-oldy);
                          if (trans_choice == 2) dyt = dyt + (y-oldy);
                          if (trans_choice == 3) dyz = dyz + (y-oldy);
                                  redraw_needed = TRUE;
                        }
                    }   /*  end while exitflag FALSE
                                  and (qtest or redraw_needed FALSE) */
                }   /*  end while exitflag FALSE */
                exit(0);
}   /*  end main()  */


/*                  printhits
 *        Prints wire parameter data in data window
 */
void printhits(iblink)
long iblink;
{
    int i,j;
    char str[60];
    i = iblink-1;
    j = jwire[i]-1;
    cmov(0.2,3.6,0.0);
    sprintf(str, "Model contains %ld wires, with %ld segments.", nwires, nsegs);
    charstr(str);
    cmov(0.2,3.35,0.0);
    sprintf(str, "You have selected wire %ld, segment %ld", jwire[i], iblink);
    charstr(str);
    cmov(0.2,3.1,0.0);
    sprintf(str, "   (this is segment %ld of %ld on TAG number %ld).",
            jseg[i],isegs[j],itags[j]);
    charstr(str);
```

```
    cmov(0.2,2.85,0.0);
    sprintf(str, "Wire endpoints are: (%.2f, %.2f, %.2f),",
            bgnwire[j][0],bgnwire[j][1],bgnwire[j][2]);
    charstr(str);
    cmov(0.2,2.6,0.0);
    sprintf(str, "                    (%.2f, %.2f, %.2f).",
            endwire[j][0],endwire[j][1],endwire[j][2]);
    charstr(str);
    cmov(0.2,2.35,0.0);
    sprintf(str, "Segment endpoints are: (%.2f, %.2f, %.2f),",
            bgnnode[i][0],bgnnode[i][1],bgnnode[i][2]);
    charstr(str);
    cmov(0.2,2.1,0.0);
    sprintf(str, "                       (%.2f, %.2f, %.2f).",
            endnode[i][0],endnode[i][1],endnode[i][2]);
    charstr(str);
    cmov(0.2,1.85,0.0);
    sprintf(str, "Segment Length = %.2f meters", segs[i]);
    charstr(str);
    cmov(0.2,1.6,0.0);
    sprintf(str, "Segment Radius = %.3f meters", rads[i]);
    charstr(str);
    cmov(0.2,1.35,0.0);
    sprintf(str, "Segment to Radius Ratio = %.2f", srrat[i]);
    charstr(str);
    cmov(0.2,1.1,0.0);
    if (data[i][3] == 0)
       sprintf(str, "Both ends of segment are connected.");
    if (data[i][3] == 2)
       sprintf(str, "One end of segment is connected.");
    if (data[i][3] == 4)
       sprintf(str, "Neither end of segment is connected.");
    charstr(str);
    cmov(0.2,0.85,0.0);
    sprintf(str, "Frequency (MHz): %.2f",freq);
    charstr(str);
    cmov(0.2,0.6,0.0);
    sprintf(str, "Current on segment (real,imag): (%.4f, %.4f)",
                 rcr[i],icr[i]);
    charstr(str);
    cmov(0.2,0.35,0.0);
    sprintf(str, "Current on segment (mag,phase): (%.4f, %.4f)",
                 mcr[i],xdata[i]);
    charstr(str);
}

/*  hsv2rgb converts from hsv to rgb colors.  It assumes that
 *  s and v both equal 1.  This can be easily changed in the
 *  future.  Result is put into global variable rgbcol[3].
 */
void hsv2rgb(float hue)
{
          float h,s,v,f,p,q,t;
          int j;

          s = 1.0;
          v = 1.0;
          h = hue/60;
          j = h;
          f = h-j;
          p = v*(1-s);
          q = v*(1-(s*f));
          t = v*(1-(s*(1-f)));
          switch (j)
          {
            case 0:
              rgbcol[0]=v*255;
              rgbcol[1]=t*255;
              rgbcol[2]=p*255;
              break;
            case 1:
              rgbcol[0]=q*255;
              rgbcol[1]=v*255;
              rgbcol[2]=p*255;
              break;
            case 2:
              rgbcol[0]=p*255;
              rgbcol[1]=v*255;
              rgbcol[2]=t*255;
              break;
            case 3:
              rgbcol[0]=p*255;
              rgbcol[1]=q*255;
              rgbcol[2]=v*255;
              break;
            case 4:
              rgbcol[0]=t*255;
              rgbcol[1]=p*255;
              rgbcol[2]=v*255;
              break;
            case 5:
              rgbcol[0]=v*255;
              rgbcol[1]=p*255;
```

```
            rgbcol[2]=q*255;
            break;
        }
}


/*                  drawscene
 *      drawscene performs all drawing.  Scene is cleared and
 *      entire scene is redrawn each time called.  Mouse input
 *      is used to change polarview viewing transformation.
 *
 *      The position of the cube is fixed.
 */

void
drawscene(short dxr, short dxt, short dxz,
        short dyr, short dyt, short dyz,
        long data_choice, long trans_choice, long iblink,
        short buffer[], long hits, Boolean picking, long do_win)
{
        short azimuth, incidence, fovy;
      float cdr, azi, inc, hue, xstep;
      float valmax, valmin;
      int i;
      unsigned precision;
        static long whitecol[] = { 255, 255, 255 };
      static long blackcol[] = { 0, 0, 0 };
      static long redcol[] = { 255, 0, 0 };
      static long orangecol[] = { 255, 63, 0 };
      static long yellowcol[] = { 255, 255, 0 };
      static long greencol[] = { 0, 255, 0 };
      static long cyancol[] = { 0, 255, 255 };
      static long bluecol[] = { 0, 0, 255 };
      static long newbluecol[] = { 64, 0, 64 };
      static long violetcol[] = { 255, 0, 255 };
      static long greycol[] = { 192, 192, 192 };
      static float vbox1[3] = { 0.1, 0.2, 0 };
      static float vbox2[3] = { 1.3, 0.2, 0 };
      static float vbox3[3] = { 1.3, 1.1, 0 };
      static float vbox4[3] = { 0.1, 1.1, 0 };
      static float vbox5[3] = { 1.3, 0.5, 0 };
      static float vbox6[3] = { 0.1, 0.5, 0 };
      static float vbox7[3] = { 1.6, 0.2, 0 };
      static float vbox8[3] = { 1.6, 0.7, 0 };
      static float vbox9[3] = { 0.1, 0.7, 0 };
      static float v1[3] = { 0.2, 0.9, 0 };
      static float v2[3] = { 0.4, 0.9, 0 };
      static float v3[3] = { 0.2, 0.8, 0 };
      static float v4[3] = { 0.4, 0.8, 0 };
      static float v5[3] = { 0.2, 0.7, 0 };
      static float v6[3] = { 0.4, 0.7, 0 };
      static float v7[3] = { 0.2, 0.6, 0 };
      static float v8[3] = { 0.4, 0.6, 0 };
      static float v9[3] = { 0.2, 0.5, 0 };
      static float v10[3] = { 0.4, 0.5, 0 };
      static float v11[3] = { 0.2, 0.4, 0 };
      static float v12[3] = { 0.4, 0.4, 0 };
      static float v13[3] = { 0.2, 0.3, 0 };
      static float v14[3] = { 0.4, 0.3, 0 };
        char text[80];

      winset(do_win);
        c3i(blackcol);
        if (picking == FALSE) clear();

      cdr = M_PI/1800.0;
      fovy = 450 + dyz;
      if (fovy < 10)
              fovy = 10;
      else if (fovy > 1790)
              fovy = 1790;

        azimuth = dxr * 5;
        incidence = dyr * 5;

      if (trans_choice == 2)
      {
          azi = cdr*azimuth;
          inc = cdr*incidence;
          tx = (dxt*fcos(azi) - dyt*fcos(inc)*fsin(azi))/100;
          ty = (dxt*fsin(azi) + dyt*fcos(inc)*fcos(azi))/100;
          tz = (dyt*fsin(inc))/100;
      }

        perspective(fovy, 5.0/4.0, 1.0, 9.0);

      if (picking == FALSE) czclear(0x000000, zval);
        pushmatrix(); /* save ModelView matrix by pushing and duplicating */
                polarview(5.0, azimuth, incidence, 0);
                /* viewing commands premultiply the ModelView matrix */
                c3i(whitecol);
              pushmatrix();
                translate(tx,ty,tz);
```

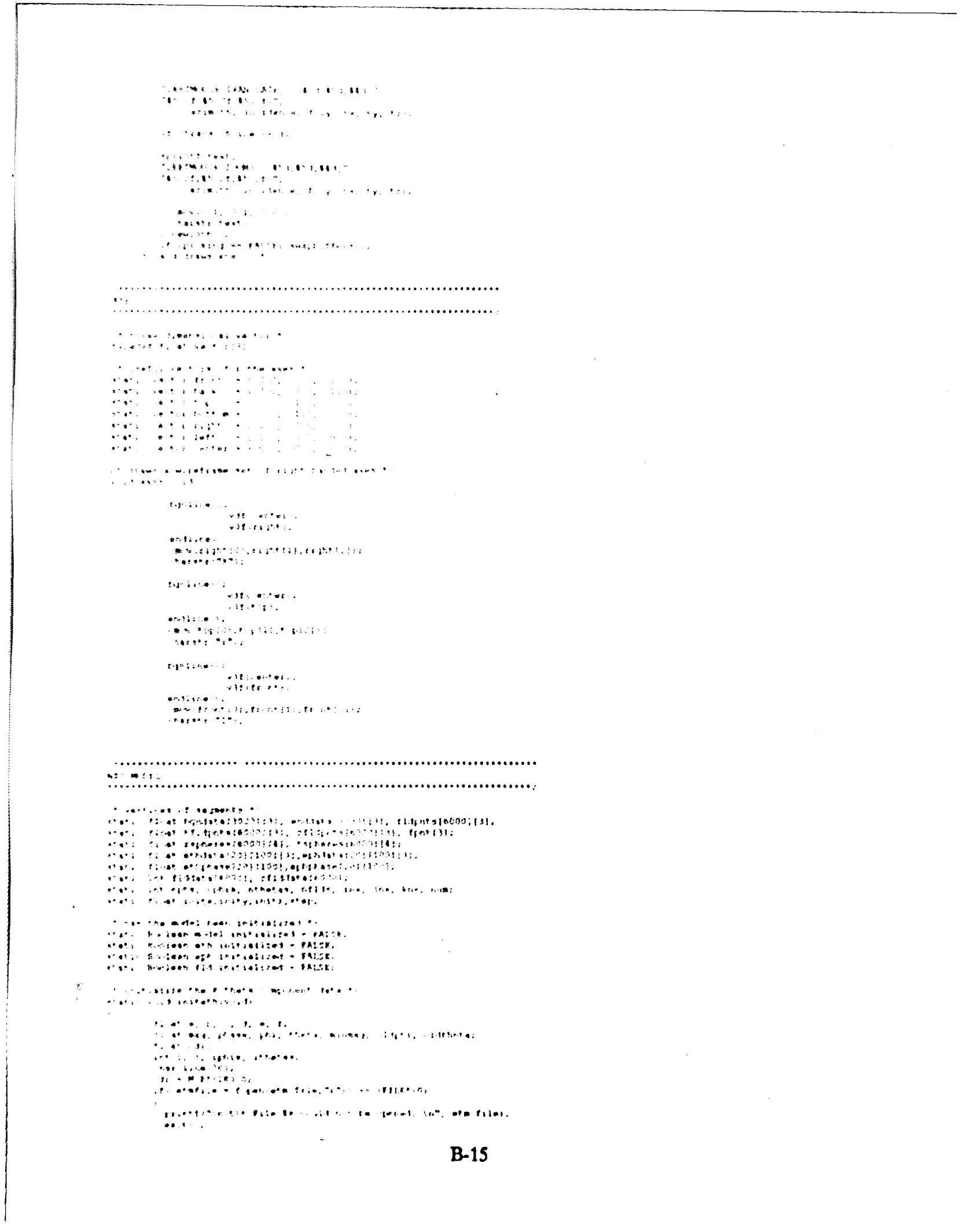
```
                wmodel(data_choice, iblink);
            popmatrix();
            c3i(greycol);
                if ((picking == FALSE)&&(draw_axes == TRUE)) axes();
          popmatrix(); /* restore ModelView matrix by popping */

          /* change projection to ortho2 to display text in 2D
           */
          ortho2(0.0, 5.0, 0.0, 4.0);
        linewidth (2);
        if (data_choice == 4)
        {
        /* make wire connections key */
/*
          bgnclosedline();
            v3f(vbox1);
            v3f(vbox7);
            v3f(vbox8);
            v3f(vbox9);
          endclosedline();
*/
          sprintf(text,
          "WIRE CONNECTIONS");
          cmov(0.2, 0.6, 0.0);
          charstr(text);
          c3i(newbluecol);
          bgnline();
            v3f(v9);
            v3f(v10);
          endline();
          c3i(whitecol);
          sprintf(te
          "Both ends c   ected");
          cmov(0 ,  , 0.0);
          char  r  );
          c3i( wncol);
          b  line();
            v3f(v11);
            v3f(v12);
          endline();
          c3i(whitecol);
          sprintf(text,
          "One end connected");
          cmov(0.5, 0.4, 0.0);
          charstr(text);
          c3i(orangecol);
          bgnline();
            v3f(v13);
            v3f(v14);
          endline();
          c3i(whitecol);
          sprintf(text,
          "Neither end connected");
          cmov(0.5, 0.3, 0.0);
          charstr(text);
        }
        else
        {
        if ((data_choice < 8)||(data_choice == 11)||(data_choice == 12))
        {
          precision = 2;
          if (data_choice == 1)
          {
            valmin = segmin;
            valmax = segmax;
          }
          if (data_choice == 2)
          {
            precision = 3;
            valmin = radmin;
            valmax = radmax;
          }
          if (data_choice == 3)
          {
            valmin = srmin;
            valmax = srmax;
            precision = 0;
          }
          if (data_choice == 5)
          {
            valmin = rcrmin;
            valmax = rcrmax;
          }
          if (data_choice == 6)
          {
            valmin = icrmin;
            valmax = icrmax;
          }
          if (data_choice == 7)
          {
            valmin = mcrmin;
            valmax = mcrmax;
          }
```

```
        if (data_choice == 11)
        {
          valmin = 0.;
          valmax = zfldmax;
        }
        if (data_choice == 12)
        {
          valmin = 0.;
          valmax = fldmax;
        }
        if (valmin == valmax)
        {
/*
          bgnclosedline();
            v3f(vbox1);
            v3f(vbox2);
            v3f(vbox5);
            v3f(vbox6);
          endclosedline();
*/
          if (data_choice == 1) sprintf(text,"SEGMENT LENGTH, METERS");
          if (data_choice == 2) sprintf(text,"WIRE RADIUS, METERS");
          if (data_choice == 3) sprintf(text,"SEGMENT TO RADIUS RATIO");
          if (data_choice == 5) sprintf(text,"REAL COMPONENT OF CURRENT");
          if (data_choice == 6) sprintf(text,"IMAG COMPONENT OF CURRENT");
          if (data_choice == 7) sprintf(text,"MAGNITUDE OF CURRENT");
          if (data_choice == 11) sprintf(text,"Z-COMP OF NEAR FIELD, VOLTS/M");
          if (data_choice == 12) sprintf(text,"TOTAL NEAR FIELD, VOLTS/M");
          cmov(0.2, 0.4, 0.0);
          charstr(text);
          c3i(newbluecol);
          bgnline();
            v3f(v13);
            v3f(v14);
          endline();
          c3i(whitecol);
          sprintf(text,
          "%6.*f", precision, valmin);
          cmov(0.5, 0.3, 0.0);
          charstr(text);
        }
        else
        {
/*
       bgnclosedline();
         v3f(vbox1);
         v3f(vbox2);
         v3f(vbox3);
         v3f(vbox4);
       endclosedline();
*/
       if (data_choice == 1) sprintf(text,"SEGMENT LENGTH, METERS");
       if (data_choice == 2) sprintf(text,"WIRE RADIUS, METERS");
       if (data_choice == 3) sprintf(text,"SEGMENT TO RADIUS RATIO");
       if (data_choice == 5) sprintf(text,"REAL COMPONENT OF CURRENT");
       if (data_choice == 6) sprintf(text,"IMAG COMPONENT OF CURRENT");
       if (data_choice == 7) sprintf(text,"MAGNITUDE OF CURRENT");
       if (data_choice == 11) sprintf(text,"Z-COMP OF NEAR FIELD, VOLTS/M");
       if (data_choice == 12) sprintf(text,"TOTAL NEAR FIELD, VOLTS/M");
       cmov(0.2, 1.0, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin, precision,
         valmin+(valmax-valmin)/7);
       cmov(0.5, 0.9, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin+(valmax-valmin)/7,
          precision, valmin+2*(valmax-valmin)/7);
       cmov(0.5, 0.8, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin+2*(valmax-valmin)/7,
          precision, valmin+3*(valmax-valmin)/7);
       cmov(0.5, 0.7, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin+3*(valmax-valmin)/7,
          precision, valmin+4*(valmax-valmin)/7);
       cmov(0.5, 0.6, 0.0);
       charstr(text);
         sprintf(text,
       "%6.*f to %6.*f", precision, valmin+4*(valmax-valmin)/7,
          precision, valmin+5*(valmax-valmin)/7);
       cmov(0.5, 0.5, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin+5*(valmax-valmin)/7,
          precision, valmin+6*(valmax-valmin)/7);
       cmov(0.5, 0.4, 0.0);
       charstr(text);
       sprintf(text,
       "%6.*f to %6.*f", precision, valmin+6*(valmax-valmin)/7,
```

```
            precision, valmin+(valmax-valmin));
        cmov(0.5, 0.3, 0.0);
        charstr(text);
        if (data_choice < 8)
        {
        c3i(newbluecol);
        bgnline();
          v3f(v1);
          v3f(v2);
        endline();
        c3i(cyancol);
        bgnline();
          v3f(v3);
          v3f(v4);
        endline();
        c3i(greencol);
        bgnline();
          v3f(v5);
          v3f(v6);
        endline();
        c3i(yellowcol);
        bgnline();
          v3f(v7);
          v3f(v8);
        endline();
        c3i(orangecol);
        bgnline();
          v3f(v9);
          v3f(v10);
        endline();
        c3i(redcol);
        bgnline();
          v3f(v11);
          v3f(v12);
        endline();
        c3i(whitecol);
        bgnline();
          v3f(v13);
          v3f(v14);
        endline();
        c3i(whitecol);
        }
        else
/*  near fields color key */
        {
        c3i(newbluecol);
        rectf(0.2, 0.9, 0.4, 0.95);
        c3i(cyancol);
        rectf(0.2, 0.8, 0.4, 0.85);
        c3i(greencol);
        rectf(0.2, 0.7, 0.4, 0.75);
        c3i(yellowcol);
        rectf(0.2, 0.6, 0.4, 0.65);
        c3i(orangecol);
        rectf(0.2, 0.5, 0.4, 0.55);
        c3i(redcol);
        rectf(0.2, 0.4, 0.4, 0.45);
        c3i(whitecol);
        rectf(0.2, 0.3, 0.4, 0.35);
        c3i(whitecol);
        }
        }
        }
        if (data_choice == 11)
        {
          if (zthresh <= 6)
          {
          sprintf(text,"THRESHOLD CUT AT %6.2f VOLTS/M", zthresh*valmax/7.);
          }
          else if (zthresh == 7)
          {
          sprintf(text,"THRESHOLD CUT AT HERP (%6.2f VOLTS/M)", herp);
          }
          else if (zthresh == 8)
          {
          sprintf(text,"THRESHOLD CUT AT HERO (%6.2f VOLTS/M)", hero);
          }
          cmov(0.2, 3.85, 0.0);
          charstr(text);
          sprintf(text,"CLICK ON MIDDLE MOUSE TO CHANGE THRESHOLD");
          cmov(0.2, 3.7, 0.0);
          charstr(text);
          sprintf(text,"FREQUENCY = %.1f MHz", freq);
          cmov(0.2, 3.55, 0.0);
          charstr(text);
        }
        if (data_choice == 12)
        {
          if (tthresh <= 6)
          {
          sprintf(text,"THRESHOLD CUT AT %6.2f VOLTS/M", tthresh*valmax/7.);
          }
          else if (tthresh == 7)
```

```
  }
  sprintf(text,"THRESHOLD CUT AT HERP (%6.2f VOLTS/M)", herp);
  }
  else if (tthresh == 8)
  {
  sprintf(text,"THRESHOLD CUT AT HERO (%6.2f VOLTS/M)", hero);
  }
  cmov(0.2, 3.85, 0.0);
  charstr(text);
  sprintf(text,"CLICK ON MIDDLE MOUSE TO CHANGE THRESHOLD");
  cmov(0.2, 3.7, 0.0);
  charstr(text);
  sprintf(text,"FREQUENCY = %.1f MHz", freq);
  cmov(0.2, 3.55, 0.0);
  charstr(text);
}
if (data_choice <= 8)
{
  sprintf(text,"USE MIDDLE MOUSE TO SELECT A WIRE SEGMENT");
  cmov(0.2, 3.85, 0.0);
  charstr(text);
}
if ((data_choice >= 5)&&(data_choice <= 8))
{
  sprintf(text,"FREQUENCY = %.1f MHz", freq);
  cmov(0.2, 3.7, 0.0);
  charstr(text);
}
if ((data_choice == 8)||(data_choice == 9)||(data_choice == 10))
{
  if (data_choice == 9)
  {
    sprintf(text,"MAX MAGNITUDE: %6.2f", ethmax);
    cmov(0.2, 3.85, 0.0);
    charstr(text);
    sprintf(text,"FREQUENCY = %.1f MHz", freq);
    cmov(0.2, 3.7, 0.0);
    charstr(text);
  }
  if (data_choice == 10)
  {
    sprintf(text,"MAX MAGNITUDE: %6.2f", ephmax);
    cmov(0.2, 3.85, 0.0);
    charstr(text);
    sprintf(text,"FREQUENCY = %.1f MHz", freq);
    cmov(0.2, 3.7, 0.0);
    charstr(text);
  }
  sprintf(text,"PHASE");
  cmov(0.2, 1.0, 0.0);
  charstr(text);
  for (i=0; i<30; i++)
  {
    hue = 360. - i*6.-3.;
    hsv2rgb(hue);
    c3i(rgbcol);
    xstep = 0.2 + i * .03;
    rectf(xstep, 0.7, xstep+.03, 0.85);
  }
  for (i=0; i<30; i++)
  {
    hue = 180. - i*6.-3.;
    hsv2rgb(hue);
    c3i(rgbcol);
    xstep = 0.2 + i * .03;
    rectf(xstep,0.5,xstep+.03,0.65);
  }
  c3i(whitecol);
  sprintf(text,"-180");
  cmov(0.2,0.9,0.0);
  charstr(text);
  sprintf(text,"0");
  cmov(0.2,0.4,0.0);
  charstr(text);
  sprintf(text,"0");
  cmov(1.06,0.9,0.0);
  charstr(text);
  sprintf(text,"180");
  cmov(0.98,0.4,0.0);
  charstr(text);
}
}
c3i(whitecol);
if (trans_choice == 1)
{
sprintf(text,
"LEFTMOUSE ROTATES: (%5d,%5d,%4d,"
"%5.2f,%5.2f,%5.2f)",
     azimuth, incidence, fovy, tx, ty, tz);
}
if (trans_choice == 2)
{
sprintf(text,
```



```
        }
        if((etpfile = fopen(etp_file,"r")) == (FILE*)0)
        {
          printf("\n\t\t File %s could not be opened. \n", etp_file);
          exit();
        }

        i = 0;
        iphis = 0;
        ithetas = 0;
        fscanf(etmfile, "%s", line);
        fscanf(etmfile, "%s", line);
        fscanf(etmfile, "%f , %f , %f , %f , %f , %f",
                &minmag, &ethmax, &a, &b, &c, &d);
        fscanf(etmfile, "%f , %f , %f", &mag, &phi, &theta);
        oldtheta = theta;
        oldphi = phi;
        while ((mag > -1.23)||(mag < -1.24)||(phi > -1.23)||(phi < -1.24)
                ||(theta > -1.23)||(theta < -1.24))
        {
          if (theta > oldtheta) ithetas++;
          if (phi > oldphi) iphis++;
          if (theta < oldtheta) ithetas = 0;
          if (phi < oldphi) iphis = 0;
          ethdata[iphis][ithetas][0] = 2*mag*fcos(cdr*theta)*fsin(cdr*phi)/ethmax;
          ethdata[iphis][ithetas][1] = 2*mag*fsin(cdr*theta)*fsin(cdr*phi)/ethmax;
          ethdata[iphis][ithetas][2] = 2*mag*fcos(cdr*phi)/ethmax;
          i++;
          oldtheta = theta;
          oldphi = phi;
          fscanf(etmfile, "%f , %f , %f", &mag, &phi, &theta);
        }
        fclose(etmfile);
        npts = i;
        nphis = iphis + 1;
        nthetas = ithetas + 1;
        fscanf(etpfile, "%s", line);
        fscanf(etpfile, "%s", line);
        fscanf(etpfile, "%f , %f , %f , %f , %f , %f", &a, &b, &c, &d, &e, &f);
        for (i=0; i < nthetas; i++)
        {
          for (j=0; j < nphis; j++)
          {
            fscanf(etpfile, "%f , %f , %f", &phase, &phi, &theta);
/* put phase between 0 and 360 */
            while (phase < 0.0) phase = phase + 360;
            while (phase > 360.0) phase = phase - 360;
            ethphase[j][i] = phase;
          }
        }
        fclose(etpfile);
        eth_initialized = TRUE;
}

/* initialize the E-phi component data */
static void initeph(void)
{
        float a, b, c, d, e, f;
        float mag, phase, phi, theta, minmag, oldphi, oldtheta;
        float cdr;
        int i, j, iphis, ithetas;
        char line[70];
        cdr = M_PI/180.0;
        if((epmfile = fopen(epm_file,"r")) == (FILE*)0)
        {
          printf("\n\t\t File %s could not be opened. \n", epm_file);
          exit();
        }
        if((eppfile = fopen(epp_file,"r")) == (FILE*)0)
        {
          printf("\n\t\t File %s could not be opened. \n", epp_file);
          exit();
        }

        i = 0;
        iphis = 0;
        ithetas = 0;
        fscanf(epmfile, "%s", line);
        fscanf(epmfile, "%s", line);
        fscanf(epmfile, "%f , %f , %f , %f , %f , %f",
                &minmag, &ephmax, &a, &b, &c, &d);
        fscanf(epmfile, "%f , %f , %f", &mag, &phi, &theta);
        oldtheta = theta;
        oldphi = phi;
        while ((mag > -1.23)||(mag < -1.24)||(phi > -1.23)||(phi < -1.24)
                ||(theta > -1.23)||(theta < -1.24))
        {
          if (theta > oldtheta) ithetas++;
          if (phi > oldphi) iphis++;
          if (theta < oldtheta) ithetas = 0;
          if (phi < oldphi) iphis = 0;
          ephdata[iphis][ithetas][0] = 2*mag*fcos(cdr*theta)*fsin(cdr*phi)/ephmax;
          ephdata[iphis][ithetas][1] = 2*mag*fsin(cdr*theta)*fsin(cdr*phi)/ephmax;
```

```
            ephdata[iphis][ithetas][2] = 2*mag*fcos(cdr*phi)/ephmax;
            i++;
            oldtheta = theta;
            oldphi = phi;
            fscanf(epmfile, "%f , %f , %f", &mag, &phi, &theta);
          }
          fclose(epmfile);
          npts = i;
          nphis = iphis + 1;
          nthetas = ithetas + 1;
          fscanf(eppfile, "%s", line);
          fscanf(eppfile, "%s", line);
          fscanf(eppfile, "%f , %f , %f , %f , %f", &a, &b, &c, &d, &e, &f);
          for (i=0; i < nthetas; i++)
          {
            for (j=0; j < nphis; j++)
            {
            fscanf(eppfile, "%f , %f , %f", &phase, &phi, &theta);
/* put phase between 0 and 360 */
            while (phase < 0.0) phase = phase + 360;
            while (phase > 360.0) phase = phase - 360;
            ephphase[j][i] = phase;
            }
          }
          fclose(eppfile);
          eph_initialized = TRUE;
}

/* initialize the near field data */
static void initfld(void)
{
          float x, y, z, zfld, fld;
          int i, j;
          if((fldfile = fopen(fld_file,"r")) == (FILE*)0)
          {
            printf("\n\t\t File %s could not be opened. \n", fld_file);
            exit();
          }
          zfldmax = 0.;
          fldmax = 0.;
          fscanf(fldfile, "%d", &nflds);
          for (i=0; i < nflds; i++)
          {
            fscanf(fldfile, "%f %f %f %f %f", &x, &y, &z, &zfld, &fld);
            if (zfld > zfldmax) zfldmax = zfld;
            if (fld > fldmax) fldmax = fld;
            fldpnts[i][0] = (x-xshift)*mscale;
            fldpnts[i][1] = (y-yshift)*mscale;
            fldpnts[i][2] = (z-zshift)*mscale;
            zflddata[i] = zfld*100.;
            flddata[i] = fld*100.;
          }
          for (i=0; i < nflds; i++)
          {
            tfldpnts[i][0] = fldpnts[i][0];
            tfldpnts[i][1] = fldpnts[i][1];
            tfldpnts[i][2] = fldpnts[i][2]+flddata[i]/fldmax/100.*mscale;
            zfldpnts[i][0] = fldpnts[i][0];
            zfldpnts[i][1] = fldpnts[i][1];
            zfldpnts[i][2] = fldpnts[i][2]+zflddata[i]/zfldmax/100.*mscale;
            zspheres[i][0] = fldpnts[i][0];
            zspheres[i][1] = fldpnts[i][1];
            zspheres[i][2] = fldpnts[i][2];
            tspheres[i][0] = fldpnts[i][0];
            tspheres[i][1] = fldpnts[i][1];
            tspheres[i][2] = fldpnts[i][2];
            zspheres[i][3] = zflddata[i]/zfldmax/400.*mscale;
            tspheres[i][3] = flddata[i]/fldmax/400.*mscale;
          }
          fclose(fldfile);
          fld_initialized = TRUE;
}

/* initialize the models vertices */
static void initmodel(void)
{
           float a, b, c, d, e, f, xpcr;
           int i, itag, iseg, irad, isr, icon, ircr, iicr, imcr, iw, is;

          fscanf(geofile, "%f", &freq);
          if (freq < 3.) herp = 632.5;
          else if (freq < 30.) herp = 1897./freq;
          else if (freq < 300.) herp = 63.25;
          else if (freq < 1500.) herp = 3.65 * sqrt(freq);
          else herp = 141.4;
          if (freq < 1.) hero = 100./freq;
          else if (freq < 3.7) hero = 100./pow(freq,3.);
          else if (freq < 10.) hero = 2.;
          else hero = 0.5*pow(freq,0.6);
            fscanf(geofile, "%d %d", &nwires, &nseqs);
          fscanf(geofile, "%f %f", &mxmin, &mxmax);
          fscanf(geofile, "%f %f", &mymin, &mymax);
          fscanf(geofile, "%f %f", &mzmin, &mzmax);
```

```
        fscanf(geofile, "%f %f", &xshift, &yshift);
        fscanf(geofile, "%f %f", &zshift, &mscale);
        fscanf(geofile, "%f %f", &segmin, &segmax);
        fscanf(geofile, "%f %f", &radmin, &radmax);
        fscanf(geofile, "%f %f", &srmin, &srmax);
        fscanf(geofile, "%f %f", &rcrmin, &rcrmax);
        fscanf(geofile, "%f %f", &icrmin, &icrmax);
        fscanf(geofile, "%f %f", &mcrmin, &mcrmax);

        for (i = 0; i < nwires; i++)
          {
          fscanf(geofile, "%d %d %f %f %f %f %f %f",&itag, &iseg, &a,
                 &b, &c, &d, &e, &f);
          itags[i] = itag;
          isegs[i] = iseg;
          bgnwire[i][0] = a;
            bgnwire[i][1] = b;
            bgnwire[i][2] = c;
          endwire[i][0] = d;
          endwire[i][1] = e;
          endwire[i][2] = f;
          }

          for (i = 0; i < nsegs; i++)
            {
            fscanf(geofile, "%d %d %d %d %d %d %d %d %d %d %f", &is, &iw,
           &iseg, &irad, &isr, &icon,
           &ircr, &iicr, &imcr, &xpcr);
          jseg[i] = is;
          jwire[i] = iw;
          data[i][0] = iseg;
          data[i][1] = irad;
          data[i][2] = isr;
          data[i][3] = icon;
          data[i][4] = ircr;
          data[i][5] = iicr;
          data[i][6] = imcr;
          xdata[i] = xpcr;
          fscanf(geofile, "%f %f %f %f %f %f", &a, &b, &c, &d, &e, &f);
          segs[i] = a;
          rads[i] = b;
          srrat[i] = c;
          rcr[i] = d;
          icr[i] = e;
          mcr[i] = f;
          fscanf(geofile, "%f %f %f %f %f %f", &a, &b, &c, &d, &e, &f);
          bgnnode[i][0] = a;
            bgnnode[i][1] = b;
            bgnnode[i][2] = c;
          endnode[i][0] = d;
          endnode[i][1] = e;
          endnode[i][2] = f;
          bgndata[i][0] = (a-xshift)*mscale;
          bgndata[i][1] = (b-yshift)*mscale;
          bgndata[i][2] = (c-zshift)*mscale;
          enddata[i][0] = (d-xshift)*mscale;
          enddata[i][1] = (e-yshift)*mscale;
          enddata[i][2] = (f-zshift)*mscale;
            }

          fclose(geofile);
          model_initialized = TRUE;
}

/* wireframe NEC model */
void wmodel(long data_choice, long iblink)
{
          int i, j;
        float h,s,v,f,p,q,t,val;
        static float origin[] = { 0, 0, 0 };
        static long redcol[] = { 255, 0, 0 };
        static long orangecol[] = { 255, 63, 0 };
        static long yellowcol[] = { 255, 255, 0 };
        static long greencol[] = { 0, 255, 0 };
        static long cyancol[] = { 0, 255, 255 };
        static long bluecol[] = { 0, 0, 255 };
        static long newbluecol[] = { 64, 0, 64 };
        static long violetcol[] = { 255, 0, 255 };
        static long greycol[] = { 192, 192, 192 };
        static long purplcol[] = { 255, 0, 176 };
        static long whitecol[] = { 255, 255, 255 };
        linewidth(2);
          if (! model_initialized)
          initmodel();
        if ((! eth_initialized)&&(data_choice == 9))
          initeth();
        if ((! eph_initialized)&&(data_choice == 10))
          initeph();
        if ((! fld_initialized)&&((data_choice == 11)||(data_choice == 12)))
          initfld();

          /* draw segments */
        if (data_choice < 9)
```

```
{
  for (i = 0; i < nsegs; i++)
    {
  if (data_choice < 8)
  {
  if (data[i][data_choice-1] == 0)
         c3i(newbluecol);
  else if (data[i][data_choice-1] == 1)
         c3i(cyancol);
  else if (data[i][data_choice-1] == 2)
         c3i(greencol);
  else if (data[i,[data_choice-1] == 3)
         c3i(yellowcol);
  else if (data[i][data_choice-1] == 4)
         c3i(orangecol);
  else if (data[i][data_choice-1] == 5)
         c3i(redcol);
  else if (data[i][data_choice-1] == 6)
         c3i(whitecol);
  }
  if (data_choice == 8)
  {
  hsv2rgb(xdata[i]);
  c3i(rgbcol);
  }
  loadname(i+1);
  if (i != iblink-1)
  {
    bgnline();
      v3f(bgndata[i]);
      v3f(enddata[i]);
    endline();
  }
    }
  }
  if (data_choice == 9)
  {
  for (i = 0; i < nphis-1; i++)
  {
    bgnqstrip();
    for (j = 0; j < nthetas; j++)
    {
    hsv2rgb(ethphase[i][j]);
    c3i(rgbcol);
    v3f(ethdata[i][j]);
    hsv2rgb(ethphase[i+1][j]);
    c3i(rgbcol);
    v3f(ethdata[i+1][j]);
    }
    endqstrip();
  }
  }
  if (data_choice == 10)
  {
  for (i = 0; i < nphis-1; i++)
  {
    bgnqstrip();
    for (j = 0; j < nthetas; j++)
    {
    hsv2rgb(ephphase[i][j]);
    c3i(rgbcol);
    v3f(ephdata[i][j]);
    hsv2rgb(ephphase[i+1][j]);
    c3i(rgbcol);
    v3f(ephdata[i+1][j]);
    }
    endqstrip();
  }
  }
  if (data_choice == 11)
  {
    c3i(greycol);
    for (i = 0; i < nsegs; i++)
    {
      bgnline();
        v3f(bgndata[i]);
        v3f(enddata[i]);
      endline();
    }
    for (i = 0; i < nflds; i++)
    {
      val = zflddata[i]/zfldmax*7./100.;
      if (((val >= zthresh)&&(zthresh < 7))||
         ((zflddata[i] >= herp*100.)&&(zthresh == 7))||
         ((zflddata[i] >= hero*100.)&&(zthresh == 8)))
      {
        if (val < 1.)
        {
          c3i(newbluecol);
          bgnpoint();
            v3f(fldpnts[i]);
          endpoint();
        }
```

```
            else
            {
              initx = fldpnts[i][0] - mscale/2.;
              inity = fldpnts[i][1] - mscale/2.;
              initz = fldpnts[i][2] - mscale/2.;
              if (val < 2.)
              {
                c3i(cyancol);
                num = 2;
              }
              else if (val < 3.)
              {
                c3i(greencol);
                num = 3;
              }
              else if (val < 4.)
              {
                c3i(yellowcol);
                num = 5;
              }
              else if (val < 5.)
              {
                c3i(orangecol);
                num = 7;
              }
              else if (val < 6.)
              {
                c3i(redcol);
                num = 9;
              }
              else
              {
                c3i(whitecol);
                num = 11;
              }
              step = (mscale/2.)/((num-1.)/2.);
              for (inx = 0; inx < num; inx++)
              {
                fpnt[0] = initx + inx * step;
                for (jnx = 0; jnx < num; jnx++)
                {
                  fpnt[1] = inity + jnx * step;
                  bgnpoint();
                  for (knx = 0; knx < num; knx++)
                  {
                    fpnt[2] = initz + knx * step;
                    v3f(fpnt);
                  }
                  endpoint();
                }
              }
            }
          }
          linewidth(2);
        }
        if (data_choice == 12)
        {
          c3i(greycol);
          for (i = 0; i < nsegs; i++)
          {
            bgnline();
              v3f(bgndata[i]);
              v3f(enddata[i]);
            endline();
          }
          for (i = 0; i < nflds; i++)
          {
            val = flddata[i]/fldmax*7./100.;
            if (val >= 6.) c3i(whitecol);
            else if (val >= 5.) c3i(redcol);
            else if (val >= 4.) c3i(orangecol);
            else if (val >= 3.) c3i(yellowcol);
            else if (val >= 2.) c3i(greencol);
            else if (val >= 1.) c3i(cyancol);
            else c3i(newbluecol);
            if (((val >= tthresh)&&(tthresh < 7))||
               ((flddata[i] >= herp*100.)&&(tthresh == 7))||
               ((flddata[i] >= hero*100.)&&(tthresh == 8)))
            {
/*
              bgnline();
                v3f(fldpnts[i]);
                v3f(tfldpnts[i]);
              endline();
*/
              if (val < 1.)
              {
                bgnpoint();
                  v3f(fldpnts[i]);
                endpoint();
              }
              else
                sphdraw(tspheres[i]);
            }
          }
          linewidth(2);
        }
}
```

Appendix C
SOURCE CODE FOR *view_mom*

```
/*      view_mom.c
 *      view_mom displays a method of moments matrix.
 */

#include <gl/gl.h>
#include <ql/device.h>
#include <stdio.h>
#include <math.h>
#define BUFSIZE 50


/*        function prototypes
 */
void initialize(void);
void main(void);
void printhits(long iblink, short row, short column);
void hsv2rgb(float h);
void drawscene(short dxr, short dxt, short dxz, short dxs,
               short dyr, short dyt, short dyz, short dys,
               long data_choice, long trans_choice, long display_mode, long iblink,
               long do_win);
void axes(void);
void initmatrix(void);
void mommatrix(long data_choice, long display_mode, long iblink);
FILE *momfile;
char file_name[25];
char mom_file[25];
static char mom[] = ".mom";
int  nnodesx,nnodesy;
float mommag[1000][1000], momphase[1000][1000];
float data[1000][1000][3];
float zscale, mommax, mommin;
long rgbcol[3];
Boolean picking_row = FALSE;
Boolean picking_col = FALSE;

static float tx, ty, tz;
/* window id's */
long mom_win, ph_win, cur_win, data_win;
static char mom_win_name[] = "MoM Matrix";
static char ph_win_name[] = "MoM Matrix with Phase";
static char data_win_name[] = "Pick Correlation Info";
long xmax, ymax, zval;


/*                  initialize
 *        The initialize subroutine positions the window and specifies
 *        its future constraints.  Graphics configurations are set and
 *        the event queue is initialized.
 */

void
initialize(void)
{
        float mreal, mimag, mag, phase, cdr;
        int i, j;

        printf(" Enter file name: ");
        scanf("%s", file_name);
        strcpy(mom_file,file_name);
        strcat(mom_file,mom);
        if((momfile = fopen(mom_file,"r")) == (FILE*)0)
        {
          printf("\n\t\t File %s could not be opened. \n", mom_file);
          exit();
        }
        cdr = M_PI/180.0;
        mommax = 0;
        mommin = 100000;
        fscanf(momfile, "%d %d", &nnodesx, &nnodesy);
        for (i=0; i < nnodesx; i++)
        {
          for (j=0; j < nnodesy; j++)
          {
            fscanf(momfile, " ( %f , %f , ", &mreal, &mimag);
            mag = fsqrt(mreal * mreal + mimag * mimag);
            phase = fatan2(mimag, mreal);
            phase = phase/cdr;
            if (mag > mommax) mommax = mag;
            if (mag < mommin) mommin = mag;
            mommag[i][j] = mag;
            while (phase < 0.0) phase = phase + 360;
            while (phase > 360.0) phase = phase - 360;
            momphase[i][j] = phase;
          }
        }
        for (i=0; i < nnodesx; i++)
        {
          for (j=0; j < nnodesy; j++)
```

```
        {
          data[i][j][0] = 2.0*i/(nnodesx-1) - 1.0;
          data[i][j][1] = 2.0*j/(nnodesy-1) - 1.0;
          data[i][j][2] = (flog10(mommin/mommag[i][j])/flog10(mommin/mommax));
        }
      }
      fclose(momfile);

        xmax = getgdesc(GD_XPMAX);
        ymax = getgdesc(GD_YPMAX);
      zval = getgdesc(GD_ZMAX);

        minsize(xmax/10, ymax/10);
      maxsize(xmax-200, ymax-160);
        keepaspect(xmax, ymax);
        mom_win = winopen("");
      wintitle(mom_win_name);
        winconstraints();
      winset(mom_win);
      zbuffer(TRUE);
        doublebuffer();
        RGBmode();
        gconfig();

        shademodel(GOURAUD);
        mmode(MVIEWING);

        qdevice(LEFTMOUSE);
        qdevice(MIDDLEMOUSE);
      qdevice(RIGHTMOUSE);
        qdevice(ESCKEY);
        tie(LEFTMOUSE, MOUSEX, MOUSEY);


}       /*   end initialize()   */


/*                main
 *      main calls initialize, then goes into a loop.  Within the loop
 *      drawscene is called and events in the event queue are processed
 *      until user exits (by pressing ESCAPE or through the window manager).
 *      Mouse input is passed to drawscene when the left mouse button
 *      is held down.
 */

void
main(void)
{
        Boolean exitflag = FALSE;
        Boolean redraw_needed = TRUE;
      Boolean draw_mom = TRUE;
      Boolean draw_ph = FALSE;
      Boolean draw_data = FALSE;
        short value;
      short buffer[BUFSIZE];
      long row_hits = 0;
      long col_hits = 0;
      short deltax, deltay;
      short row, column;
        long dev;
      long choice;
      long data_choice, trans_choice, display_mode, iblink;
      long menu, datamenu, transmenu, displaymenu;
        short x, oldx;            /*  mouse movement */
        short y, oldy;
        short dxr, dxt, dxz, dxs = 0;
        short dyr, dyt, dyz, dys = 0;
      static long whitecol[] = { 255, 255, 255 };
      static long bluecol[] = { 0, 0, 255 };
      static long dbluecol[] = { 0, 0, 128 };

        initialize();
      data_choice = 1;
      trans_choice = 1;
      display_mode = 2;
      iblink = 0;
      tx = 0;
      ty = 0;
      tz = 0;

      /* make the data menu */
      datamenu = defpup("Choose Data Window to Open/Close %t|"
        "Magnitude %x1|Magnitude & Phase %x2");
      /* make the transformation menu */
      transmenu = defpup("Choose Transformation %t|"
        "Rotate %x3|Translate %x4|Zoom %x5|Scale Z-comp %x6|Return to initial state %x7");
      /* make the display mode menu */
      displaymenu = defpup("Choose Display Mode %t|"
        "0D %x9|1D %x10|2D %x11");
      /* make the main menu */
      menu = defpup("Menu %t|Data Window %m|Transformation %m|Display Mode %m|"
        "Close Current Window %x8",datamenu,transmenu,displaymenu);
```

```
while (exitflag == FALSE)
{
          if (redraw_needed == TRUE)
          {
      if (draw_mom)
          drawscene(dxr, dxt, dxz, dxs, dyr, dyt, dyz, dys,
                  1, trans_choice, display_mode, iblink,
                  mom_win);
      if (draw_ph)
          drawscene(dxr, dxt, dxz, dxs, dyr, dyt, dyz, dys,
                  2, trans_choice, display_mode, iblink,
                  ph_win);

                    if (iblink == 0) redraw_needed = FALSE;
              iblink = -iblink;
          }

          while ( (exitflag == FALSE) &&
                    (qtest() || (redraw_needed == FALSE)) )
          {
                    dev = qread (&value);
                    if (dev == ESCKEY)
                    {
                              if (value == 0)

                                        exitflag = TRUE;
                    }
                    else if (dev == REDRAW)
                    {
                              reshapeviewport();
                              redraw_needed = TRUE;
                    }
              else if (dev == INPUTCHANGE)
              {
                    cur_win = value;
              }
              else if (dev == RIGHTMOUSE && value == 1)
              {
                      choice = dopup(menu);
                      if (choice >= 1)
                      {
                      if (choice == 1)
                      {
                        draw_mom = !draw_mom;
                        if (draw_mom)
                        {
                            minsize(xmax/10, ymax/10);
                          maxsize(xmax-200, ymax-160);
                            keepaspect(xmax, ymax);
                            mom_win = winopen("");
                          wintitle(mom_win_name);
                            winconstraints();
                          winset(mom_win);
                          zbuffer(TRUE);
                            doublebuffer();
                            RGBmode();
                            gconfig();
                            shademodel(GOURAUD);
                            mmode(MVIEWING);
                        }
                        else winclose(mom_win);
                      }
                      if (choice == 2)
                      {
                        draw_ph = !draw_ph;
                        if (draw_ph)
                        {
                            minsize(xmax/10, ymax/10);
                          maxsize(xmax-200, ymax-160);
                            keepaspect(xmax, ymax);
                            ph_win = winopen("");
                          wintitle(ph_win_name);
                            winconstraints();
                          winset(ph_win);
                          zbuffer(TRUE);
                            doublebuffer();
                            RGBmode();
                            gconfig();
                            shademodel(GOURAUD);
                            mmode(MVIEWING);
                        }
                        else winclose(ph_win);
                      }

                      if ((choice >= 3)&&(choice <= 7))
                         trans_choice = choice - 2;
                      if (trans_choice == 5)
                      {
                        dxr = 0;
                        dxt = 0;
                        dxz = 0;
                        dxs = 0;
                        dyr = 0;
```

```
            dyt = 0;
            dyz = 0;
            dys = 0;
            tx = 0;
            ty = 0;
            tz = 0;
            trans_choice = 1;
          }
          if (choice == 8)
          {
            if (cur_win == mom_win) draw_mom = !draw_mom;
            if (cur_win == ph_win) draw_ph = !draw_ph;
            winclose(cur_win);
          }

          if ((choice >= 9)&&(choice <= 11)) display_mode = choice - 9;

          redraw_needed = TRUE;
          }
}
      else if (dev == LEFTMOUSE)
      {
                if (value)
                /* read initial mouse x,y, which will be
                 * next 2 events because tied to leftmouse
                 */
                {
                          qread(&x);
                          qread(&y);
                        /* now listen to all mouse x,y
                         * as long as mouse down */
                          qdevice(MOUSEX);
                          qdevice(MOUSEY);
                }
                else
                /* as soon as mouse released,
                 * stop listening to mouse x,y */
                {
                          unqdevice(MOUSEX);
                          unqdevice(MOUSEY);
                }
      }
else if (dev == MIDDLEMOUSE && value == 0)
{
          /* do a pick correlation in current window */
          winset(cur_win);
          /* first find row */
          pick(buffer,BUFSIZE);
          picking_row = TRUE;
          deltax = 3;
          deltay = 3;
          picksize(deltax,deltay);
          iblink = 0;
          drawscene(dxr,dxt,dxz,dxs,dyr,dyt,dyz,dys,data_choice,trans_choice,
                    0,iblink,cur_win);
          picking_row = FALSE;
          row_hits = endpick(buffer);
          if (row_hits > 0) row = buffer[1];

          /* now find column */
          pick(buffer,BUFSIZE);
          picking_col = TRUE;
          deltax = 3;
          deltay = 3;
          picksize(deltax,deltay);
          iblink = 0;
          drawscene(dxr,dxt,dxz,dxs,dyr,dyt,dyz,dys,data_choice,trans_choice,
                    0,iblink,cur_win);
          picking_col = FALSE;
          col_hits = endpick(buffer);
          if (col_hits > 0) column = buffer[1];

          if ((row_hits > 0)&&(col_hits > 0))
          {
            if (draw_data == FALSE)
            {
              prefposition(xmax/40,xmax*9/20,ymax/40,ymax*6/40);
              data_win = winopen("");
              wintitle(data_win_name);
              winconstraints();
              winset(data_win);
              singlebuffer();
              RGBmode();
              gconfig();
              shademodel(FLAT);
              mmode(MVIEWING);
              draw_data = TRUE;
            }
            winset(data_win);
            c3i(dbluecol);
            clear();
            c3i(whitecol);
            ortho2(0.0, 5.0, 0.0, 4.0);
```

```
                        printhits(iblink, row, column);
                    }
                    else
                    {
                        if (draw_data)
                        {
                          draw_data = FALSE;
                          winclose(data_win);
                        }
                    }
                }
                else if (dev == MOUSEX)
                {
                        oldx = x;
                        x = value;
                  if (trans_choice == 1) dxr = dxr + (x-oldx);
                  if (trans_choice == 2) dxt = dxt + (x-oldx);
                  if (trans_choice == 3) dxz = dxz + (x-oldx);
                  if (trans_choice == 4) dxs = dxs + (x-oldx);
                        redraw_needed = TRUE;
                }
                else if (dev == MOUSEY)
                {
                        oldy = y;
                        y = value;
                  if (trans_choice == 1) dyr = dyr + (y-oldy);
                  if (trans_choice == 2) dyt = dyt + (y-oldy);
                  if (trans_choice == 3) dyz = dyz + (y-oldy);
                  if (trans_choice == 4) dys = dys + (y-oldy);
                        redraw_needed = TRUE;
                }
            }   /*  end while exitflag FALSE
                        and (qtest or redraw_needed FALSE) */
        }   /*  end while exitflag FALSE */
        exit(0);
}   /*  end main()  */

/*  printhits
 *  Prints row, column info in data window
 */
void printhits(long iblink, short row, short column)
{
    int i,j;
    char str[60];
    i = row;
    j = column;
    cmov(0.2,3.4,0.0);
    sprintf(str, "This matrix has %ld rows and %ld columns.", nnodesx, nnodesy);
    charstr(str);
    cmov(0.2,2.8,0.0);
    sprintf(str, "You have picked row %ld, column %ld.", row, column);
    charstr(str);
    cmov(0.2,2.2,0.0);
    sprintf(str, "The magnitude of the matrix at this point is: %.2f.", mommag[i][j]);
    charstr(str);
    cmov(0.2,1.6,0.0);
    sprintf(str, "The phase of the matrix at this point is: %.2f degrees.", momphase[i][j]);
    charstr(str);
    cmov(0.2,1.0,0.0);
    sprintf(str, "The maximum magnitude in this matrix is: %.2f.", mommax);
    charstr(str);
    cmov(0.2,0.4,0.0);
    sprintf(str, "The minimum magnitude in this matrix is: %.2f.", mommin);
    charstr(str);
}


/*  hsv2rgb converts from hsv to rgb colors.  It assumes that
 *  s and v both equal 1.  This can be easily changed in the
 *  future.  Result is put into global variable rgbcol[3].
 */
void hsv2rgb(float hue)
{
        float h,s,v,f,p,q,t;
        int j;

        s = 1.0;
        v = 1.0;
        h = hue/60;
        j = h;
        f = h-j;
        p = v*(1-s);
        q = v*(1-(s*f));
        t = v*(1-(s*(1-f)));
        switch (j)
        {
          case 0:
            rgbcol[0]=v*255;
            rgbcol[1]=t*255;
            rgbcol[2]=p*255;
            break;
          case 1:
            rgbcol[0]=q*255;
```

```
            rgbcol[1]=v*255;
            rgbcol[2]=p*255;
            break;
          case 2:
            rgbcol[0]=p*255;
            rgbcol[1]=v*255;
            rgbcol[2]=t*255;
            break;
          case 3:
            rgbcol[0]=p*255;
            rgbcol[1]=q*255;
            rgbcol[2]=v*255;
            break;
          case 4:
            rgbcol[0]=t*255;
            rgbcol[1]=p*255;
            rgbcol[2]=v*255;
            break;
          case 5:
            rgbcol[0]=v*255;
            rgbcol[1]=p*255;
            rgbcol[2]=q*255;
            break;
        }
}


/*                  drawscene
 *      drawscene performs all drawing.  Scene is cleared and
 *      entire scene is redrawn each time called.  Mouse input
 *      is used to change polarview viewing transformation.
 *
 */

void
drawscene(short dxr, short dxt, short dxz, short dxs,
        short dyr, short dyt, short dyz, short dys,
        long data_choice, long trans_choice, long display_mode, long iblink,
        long do_win)
{
        short azimuth, incidence, fovy;
      float cdr, azi, inc, hue, xstep;
      float valmax, valmin;
      int i;
      unsigned precision;
        static long whitecol[] = { 255, 255, 255 };
      static long blackcol[] = { 0, 0, 0 };
      static long redcol[] = { 255, 0, 0 };
      static long orangecol[] = { 255, 63, 0 };
      static long yellowcol[] = { 255, 255, 0 };
      static long greencol[] = { 0, 255, 0 };
      static long cyancol[] = { 0, 255, 255 };
      static long bluecol[] = { 0, 0, 255 };
      static long newbluecol[] = { 0, 52, 255 };
      static long violetcol[] = { 255, 0, 255 };
      static long greycol[] = { 192, 192, 192 };

      winset(do_win);
        c3i(blackcol);
        if ((picking_row == FALSE) && (picking_col == FALSE)) clear();

      cdr = M_PI/1800.0;
      fovy = 450 + dyz;
      if (fovy < 10)
              fovy = 10;
      else if (fovy > 1790)
              fovy = 1790;

        azimuth = dxr * 5;
        incidence = dyr * 5;

      if (trans_choice == 2)
      {
          azi = cdr*azimuth;
          inc = cdr*incidence;
          tx = (dxt*fcos(azi) - dyt*fcos(inc)*fsin(azi))/100;
          ty = (dxt*fsin(azi) + dyt*fcos(inc)*fcos(azi))/100;
          tz = (dyt*fsin(inc))/100;
      }

        perspective(fovy, 5.0/4.0, 1.0, 9.0);

      if ((picking_row == FALSE) && (picking_col == FALSE)) czclear(0x000000, zval);
        pushmatrix(); /* save ModelView matrix by pushing and duplicating */
                polarview(5.0, azimuth, incidence, 0);
                /* viewing commands premultiply the ModelView matrix */
                c3i(whitecol);
              pushmatrix();
                translate(tx,ty,tz);
                    mommatrix(data_choice, display_mode, iblink);
              popmatrix();
              c3i(greycol);
                  if ((picking_row == FALSE) && (picking_col == FALSE)) axes();
```

```
        popmatrix(); /* restore ModelView matrix by popping */

        /* change projection to ortho2 to display text in 2D
         */
        ortho2(0.0, 5.0, 0.0, 4.0);
      if ((picking_row == FALSE) && (picking_col == FALSE)) swapbuffers();
}   /*  end drawscene()  */


/*****************************************************************
AXES
*****************************************************************/

/* three dimensional vector */
typedef float vector[3];

/* useful vectors  for the axes */
static vector front  = { 0.0,  0.0,  1.0};
static vector back   = { 0.0,  0.0, -1.0};
static vector top    = { 0.0,  1.0,  0.0};
static vector bottom = { 0.0, -1.0,  0.0};
static vector right  = { 1.0,  0.0,  0.0};
static vector left   = {-1.0,  0.0,  0.0};
static vector center = { 0.0,  0.0,  0.0};

/* draws a wireframe set of right-handed axes */
void axes(void)
{
        bgnline();
                v3f(center);
                v3f(right);
        endline();
        cmov(right[0],right[1],right[2]);
        charstr("X");

        bgnline();
                v3f(center);
                v3f(top);
        endline();
        cmov(top[0],top[1],top[2]);
        charstr("Y");

        bgnline();
                v3f(center);
                v3f(front);
        endline();
        cmov(front[0],front[1],front[2]);
        charstr("Z");
}


/*********************************************************************
MoM MATRIX
*********************************************************************/

/* MoM matrix */
void mommatrix(long data_choice, long display_mode, long iblink)
{
        int i, j;
      short row, column;
      float h,s,v,f,p,q,t,val;
      static float origin[] = { 0, 0, 0 };
      static long redcol[] = { 255, 0, 0 };
      static long orangecol[] = { 255, 63, 0 };
      static long yellowcol[] = { 255, 255, 0 };
      static long greencol[] = { 0, 255, 0 };
      static long cyancol[] = { 0, 255, 255 };
      static long bluecol[] = { 0, 0, 255 };
      static long newbluecol[] = { 0, 52, 255 };
      static long violetcol[] = { 255, 0, 255 };
      static long greycol[] = { 192, 192, 192 };
      static long purplcol[] = { 255, 0, 176 };
      static long whitecol[] = { 255, 255, 255 };
      if ((picking_row == FALSE) && (picking_col == FALSE))
      {
        if (display_mode == 0)
        {
          bgnpoint();
          for (i=0; i < nnodesx; i++)
          {
            for (j=0; j < nnodesy; j++)
            {
              if (data_choice == 1) hsv2rgb(data[i][j][2]*360.0);
              if (data_choice == 2) hsv2rgb(momphase[i][j]);
              c3i(rgbcol);
              v3f(data[i][j]);
            }
          }
          endpoint();
        }
        if (display_mode == 1)
        {
          for (i=0; i < nnodesx; i++)
```

```
    {
      bgnline();
      for (j=0; j < nnodesy; j++)
      {
        if (data_choice == 1) hsv2rgb(data[i][j][2]*360.0);
        if (data_choice == 2) hsv2rgb(momphase[i][j]);
        c3i(rgbcol);
        v3f(data[i][j]);
      }
      endline();
    }
    for (j=0; j < nnodesy; j++)
    {
      bgnline();
      for (i=0; i < nnodesx; i++)
      {
        if (data_choice == 1) hsv2rgb(data[i][j][2]*360.0);
        if (data_choice == 2) hsv2rgb(momphase[i][j]);
        c3i(rgbcol);
        v3f(data[i][j]);
      }
      endline();
    }
  }
  if (display_mode == 2)
  {
    for (i=0; i < nnodesx-1; i++)
    {
      bgnqstrip();
      for (j=0; j < nnodesy; j++)
      {
        if (data_choice == 1) hsv2rgb(data[i][j][2]*360.0);
        if (data_choice == 2) hsv2rgb(momphase[i][j]);
        c3i(rgbcol);
        v3f(data[i][j]);
        if (data_choice == 1) hsv2rgb(data[i+1][j][2]*360.0);
        if (data_choice == 2) hsv2rgb(momphase[i+1][j]);
        c3i(rgbcol);
        v3f(data[i+1][j]);
      }
      endqstrip();
    }
  }
}
if (picking_row)
{
    for (i=0; i < nnodesx; i++)
    {
      loadname(i+1);
      bgnpoint();
      for (j=0; j < nnodesy; j++)
      {
        v3f(data[i][j]);
      }
      endpoint();
    }
}
if (picking_col)
{
    for (j=0; j < nnodesy; j++)
    {
      loadname(j+1);
      bgnpoint();
      for (i=0; i < nnodesx; i++)
      {
        v3f(data[i][j]);
      }
      endpoint();
    }
}
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1992	Final; Oct 91 — Sep 92

4. TITLE AND SUBTITLE
HIGH-PERFORMANCE VISUALIZATION APPLIED TO COMPUTATIONAL ELECTROMAGNETICS
Method-of-Moments on a Silicon Graphics Workstation

5. FUNDING NUMBERS
PE: 0602936N
PROJ: RV36I21
SUBPROJ: 82-ZF08-01

6. AUTHOR(S)
L. C. Russell and J. W. Rockway

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division (NRaD)
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
TR 1564

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division (NRaD)
San Diego, CA 92152-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This investigation applied advanced visualization techniques to an existing computational electromagnetic code. The goal was to demonstrate that high-performance visualization can improve the utility of computational techniques used in ship electromagnetic designs.

The code used was the Numerical Electromagnetics Code - Method of Moments (NEC-MoM). The visualization platform was a Silicon Graphics 4D workstation. Two codes were developed to do pre- and post-processing visualization on NEC-MoM.

This report lists several benefits that have resulted, or will result, from work done on this Independent Exploratory Development (IED) project.

14. SUBJECT TERMS		15. NUMBER OF PAGES
visualization high-performance computing	computational electromagnetic code method of moments	57
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
L. C. Russell	(619) 553-6132	Code 824

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0142	K. J. Campbell	(1)
Code 144	V. Ware	(1)
Code 80	K. D. Regan	(1)
Code 82	R. J. Kochanski	(1)
Code 824	J. B. Rhode	(1)
Code 824	L. C. Russell	(25)
Code 961	Archive/Stock	(6)
Code 964B	Library	(2)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development
Information Center (NARDIC)
Washington, DC 20360-5000

GIDEP Operations Center
Corona, CA 91718-8000